CrossTalk

March/April 2010 The Journal of Defense Software Engineering Vol. 23 No. 2

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE
APR 2010

2. REPORT TYPE

3. DATES COVERED
00-03-2010 to 00-04-2010

4. TITLE AND SUBTITLE
CrossTalk. The Journal of Defense Software Engineering. Volume 23, Number 2, March/April 2010

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
517 SMXS MXDEA,6022 Fir Ave,Hill AFB,UT,84056-5820

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:

a. REPORT	b. ABSTRACT	c. THIS PAGE
unclassified	**unclassified**	**unclassified**

17. LIMITATION OF ABSTRACT
Same as Report (SAR)

18. NUMBER OF PAGES
32

19a. NAME OF RESPONSIBLE PERSON

Systems Assurance: Preparation and Promise

ON THE COVER
Cover Design by
Kent Bingham

Departments

CROSSTALK

OSD (AT&L)	*Stephen P. Welby*
NAVAIR	*Jeff Schwalb*
309 SMXG	*Karl Rogers*
DHS	*Joe Jarzombek*
MANAGING DIRECTOR	*Brent Baxter*
PUBLISHER	*Kasey Thompson*
MANAGING EDITOR	*Drew Brown*
ASSOCIATE EDITOR	*Chelene Fortier-Lozancich*
ARTICLE COORDINATOR	*Marek Steed*
PHONE	(801) 775-5555
E-MAIL	stsc.customerservice@hill.af.mil
CROSSTALK ONLINE	www.stsc.hill.af.mil/crosstalk

CROSSTALK, The Journal of Defense Software Engineering is co-sponsored by the Office of the Secretary of Defense (OSD) Acquisition, Technology and Logistics (AT&L); U.S. Navy (USN); U.S. Air Force (USAF); and the U.S. Department of Homeland Security (DHS). OSD (AT&L) co-sponsor: Software Engineering and System Assurance. USN co-sponsor: Naval Air Systems Command. USAF co-sponsor: Ogden-ALC 309 SMXG. DHS co-sponsor: National Cybersecurity Division in the National Protection and Programs Directorate.

The USAF Software Technology Support Center (STSC) is the publisher of CROSSTALK, providing both editorial oversight and technical review of the journal. CROSSTALK's mission is to encourage the engineering development of software to improve the reliability, sustainability, and responsiveness of our warfighting capability.

S T S C

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail us or use the form on p. 27.

517 SMXS/MXDEA
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the CROSSTALK editorial board prior to publication. Please follow the Author Guidelines, available at <www.stsc.hill.af.mil/crosstalk/xtlkguid.pdf>. CROSSTALK does not pay for submissions. Published articles remain the property of the authors and may be submitted to other publications. Security agency releases, clearances, and public affairs office approvals are the sole responsibility of the author and their organizations.

Reprints: Permission to reprint or post articles must be requested from the author or the copyright holder and coordinated with CROSSTALK.

Trademarks and Endorsements: This Department of Defense (DoD) journal is an authorized publication for members of the DoD. Contents of CROSSTALK are not necessarily the official views of, or endorsed by, the U.S. government, the DoD, the co-sponsors, or the STSC. All product names referenced in this issue are trademarks of their companies.

CROSSTALK *Online Services:* See <www.stsc.hill.af.mil/crosstalk>, call (801) 777-0857 or e-mail <stsc.webmaster@hill.af.mil>.

Back Issues Available: Please phone or e-mail us to see if back issues are available free of charge.

Reinforcing Systems Assurance In Cyber Risk Management

Systems assurance is a matter of strategic concern to our nation's security; one the DHS takes very seriously. Fundamentally, it is the concerted effort to ensure that users have the highest level of confidence possible in their critical systems and data.

In other words, systems assurance is the integrated effort to enhance user confidence in the safety, security, reliability, availability, and maintainability of processes and products. A number of factors impact the user community's confidence, including standards setting, procedures development, regulations, and specific verification and certification criterion.

Industry and the government have been working hard and have made important progress in providing systems assurance for their customers. Collaborative efforts are advancing standards and practices for systems and software assurance.

However, building and sustaining customer confidence is difficult and in an ever-changing technological environment where such confidence is easily diminished. In fact, malicious actors have shown great adaptability in their efforts to undermine assurance efforts. This is why we must be constantly vigilant and strive to implement innovative systems assurance technologies.

It also must not be forgotten that enhanced systems assurance supports intellectual property rights protection, improved consumer trust, and more confident business operations and services. A broad spectrum of critical applications and infrastructure, from process control systems to commercial applications, depend on secure, resilient systems. That is why we must manage risks to these systems effectively and improve our capabilities and capacity to mitigate those risks.

Let me close with the observation that the most effective systems assurance efforts are "baked" into the risk management process from the very beginning of the system development life cycle. Security and resiliency must be integrated throughout the life cycle. If they are not, customers may unknowingly accept risks that could have profound financial, legal, and national security implications.

By working together—building on efforts already underway, and striving to take steps that will enhance and improve confidence in critical systems and data—we can make an impact that will better secure the nation.

I sincerely hope you enjoy this issue of CrossTalk and learn from the keen insights and recommendations contained herein.

Gregory Schaeffer
Assistant Secretary for Cybersecurity and Communications
U.S. Department of Homeland Security

Systems Assurance as a Team Sport

Robert A. Martin
The MITRE Corporation

It is time for individual procurement officers—and those who work with the vendors and software developers creating or delivering our day-to-day mission software-based systems—to contribute to, and accelerate the adoption of, standards-based cybersecurity. This article outlines several practical and straightforward steps that professionals can take to speed the adoption of the enterprise security initiatives currently under way, and to make those efforts have a greater and quicker impact.

Many exciting and important changes are happening within the marketplace of cybersecurity capabilities and the way organizations address compliance, assurance, and security. As documented in [1, 2, 3, 4], MITRE has collaborated with industry, government, and academia throughout the last 10 years. They have fostered the creation and adoption of a number of information security standards that are changing the concepts of compliance, assurance, and security in enterprises, products, and practices.

While still evolving, several of these standardization efforts have made their way into commercial solutions and government, industry, and academic use. Perhaps the most visible of these have been the:

- Common Vulnerabilities and Exposures (CVE) initiative.
- Executive Office of the President–Office of Management and Budget (EOP-OMB)-mandated (see [5, 6, 7, 8, 9]) Federal Desktop Core Configuration (FDCC) effort that leverages the Security Content Automation Protocol (SCAP).
- Consensus Audit Guidelines [10].
- Build Security In (BSI) initiative.

CVE is utilized by the majority of the vulnerability-related information providers and tool vendors. The SCAP utilizes CVE and the other mature standardization efforts to clearly define common security nomenclature and evaluation criteria for vulnerability, patch, and configuration measurement guidance; it is intended for adoption by automated tools. By specifying and measuring compliance in terms of these standardized concepts, the community is moving towards *compliant all the time* based on automated measurement. This method takes an annual or tri-annual activity that had questionable insights about the current security posture of an enterprise and makes it into a consistent way of *knowing* both how and that your enterprise is secured.

Similar to the transformation in operational security measurement (motivated by the CVE initiative, FDCC, and the SCAP) are the changes under way in assuring the resilience and code security of the software making up the applications and systems software bought and built in organizations and government entities. The BSI initiative is a software assurance effort that provides practices, tools, guidelines, rules, principles, and other resources software developers, architects, and security practitioners can use to build security into software in every phase of its development. BSI leverages the Common Weakness Enumeration (CWE) and the Common Attack Pattern Enumeration and Classification (CAPEC) efforts.

> *"While still evolving, several of these standardization efforts have made their way into commercial solutions and government, industry, and academic use."*

To measure and manage their cyber assets, an enterprise will need to employ consistent approaches that are supported by automation techniques, typically using products from a variety of different vendors. To make the finding and reporting of issues consistent and composable across different groups of practitioners and tools, there has to be a set of standard definitions of the things that are being examined, reported, and managed [4].

To reach this level of capability, the standardization has to make sense to commercial industry so that it will be adopted in baseline products and practices, and to the academic world so that research will continue to advance the state of the art in a complementary manner.

While there has been great progress in bringing standardization to some activities and tools in some areas like the CVE initiative, the SCAP/FDCC, and BSI, there is still more that individuals can do. Those buying software products, creating organizational policies related to the security and resiliency of systems, or creating security guidance and benchmarks can help us all get to these greater capabilities faster by taking the actions outlined in this article.

Procurement Guidance for Purchasing Software

As a procurement officer, you can make sure that the products being offered are compliant with the new Federal Acquisition Regulation provision [11], specifying compliance with the FDCC. Additionally, procurement officers can levy requirements on the software providers to:

- Submit a Common Platform Enumeration (CPE) name for each new release of the provider's software.
- Have a public address (e-mail and/or Internet) for the reporting of security relevant issues with the provider's software.
- Have a publicly available statement of the time frame and process that the software provider's organization follows in addressing reports of security relevant issues with the provider's software.
- Have public advisories of relevant security-related issues and their resolution.
- Include a CVE Identifier for security-related issues when the issues are related to a software flaw or default setting that constitutes a security shortcoming in the provider's software (as part of the initial public advisory).
- Include an initial Open Vulnerability and Assessment Language (OVAL) definition(s) as a machine-readable description of how to tell if the flaw,

misconfiguration, or incorrect default settings are present and whether any of the known resolutions have been taken as part of the initial public advisory.

- Include the base and initial temporal severity score portions of the Common Vulnerability Scoring System (CVSS) rating for the flaw or incorrect default settings as part of the initial public advisory.
- Provide advice to customers on how to securely configure their software and do so utilizing the standards within the SCAP. Specifically, offer eXtensible Configuration Checklist Description Format (XCCDF) and OVAL documents representing the vendor's recommendations on secure configuration. Include the appropriate Common Configuration Enumeration (CCE) identifiers for the configuration controls that they recommend settings for and CPE names for the software packages discussed.
- Identify and discuss the assurance activities that their software products go through—in terms of the CWE names that are reviewed and tested for and the CAPEC names utilized in the analysis and evaluation activities performed on their software.

Government Organizations

As a government organization decides how systems should be set-up for operational use, standards can be used to convey this *blessed* configuration. This is referred to in [12] as a SCAP-expressed checklist. Specifically, government organizations can levy requirements on their user communities to:

- Express policies and guidelines in the XCCDF/OVAL languages so that tool technologies can use these machine-readable descriptions to evaluate the status of information technology with regards to those policies and guidelines.
- Adopt automated methods utilizing the machine-readable XCCDF/OVAL policies and guidelines for assessing, reporting, and directing action on exceptions to the policies and guidelines.

Similarly, as a government organization establishes its approach to gaining assurance about the robustness, integrity, and secureness of the software its systems contain, standards can be used to clarify and specify the types of evidence and insights needed to gain sufficient software assurance. Specifically, government organizations can require their user communities to:

- Express policies and guidelines about the assurance of a capability by discussing the security weaknesses that have been vetted for in terms of CWE names and methods used to verify and test for security issues in terms of CAPEC names.
- Adopt the use of automated assessment methods that utilize CWE and CAPEC for assessing, reporting, documenting, and directing action on weaknesses and exceptions to the assurance policies and guidelines. This way, issues can be tracked and managed in a vendor-neutral and consistent manner.

Procurement Guidance for Purchasing Security Tools

In general, procurement and end-users should request or require that the vendors of security products that deal with security flaws, configuration settings, policies, or patches support the SCAP standards. It is strongly recommended that the automated tools used to implement or verify security controls employ SCAP (or similar standardization) efforts for clearly defined nomenclature and evaluation criteria not covered by the SCAP. Specifically, these types of products and services should:

- Include the appropriate CVE Identifier for security information that is related to a software flaw or a non-secure default setting.
- Provide for the searching of security-related information by CVE Identifier.
- Incorporate the machine-readable tests for flaws, patches, and configuration checks written in conformance with the OVAL Definition Schema.
- Generate machine-readable assessment results from tests for flaws, patches, and configuration checks in conformance with the XCCDF and OVAL Results Schema. In the near future, expect products to produce results in the Assessment Results Format, which is an emerging SCAP specification that describes an XML Schema for sharing per-device assessment results of devices on IP-routed networks.
- Incorporate the machine-readable results from flaw, patch, and configuration check assessments that are written in conformance with the OVAL Results Schema.
- Incorporate, as appropriate to the functionality of the tool, support for the different severity score portions of the CVSS rating for the flaw or incorrect default settings.

An *assurance ecosystem* has emerged around these various types of enumerative and language-based standardization and has been adopted in various ways by government and commercial industry[1]; it continues to provide the framework to the academic world for continued research and to industry in advancing the state of the art in a complementary, interoperable manner.

While there has been great progress in bringing standardization to many practices and tools, there is more that individuals can do to push even greater capabilities to emerge in a timely manner. Procurement officials and consumers of software products, as well as organizational security policy makers, should take more deliberate action to demand secure products and create security guidance and benchmarks, in turn helping to get to these greater capabilities faster; [12] addresses many aspects of this issue. Similar adoption endorsement initiatives and efforts—to move forward and support other useful standards initiatives—

Table 1: *Emerging Standardization Efforts*

Activity	Focus	Web site
Assessment Results Format	Result Reporting	<http://msm.mitre.org/incubator>
Common Event Expressions	Security Events	<http://cee.mitre.org>
Policy Language for Assessment Results Reporting	Result Reporting	<http://msm.mitre.org/incubator/plarr>
Malware Attribute Enumeration and Characterization	Malware Attributes	<https://buildsecurityin.us-cert.gov/swa/malact.html>
Common Remediation Enumeration	Remediation Actions	<http://msm.mitre.org/incubator>
Common Remediation Language	Remediation Actions	<http://msm.mitre.org/incubator>
Open Checklist Interactive Language	Interactive Survey Questions	<http://scap.nist.gov/specifications/ocil>
Open Checklist Reporting Language	Interactive Survey Answers	<http://ocrl.mitre.org>

Software Defense Application

The rollout of efforts like SCAP-enabled operational measurement is setting the stage for higher levels of assurance in cybersecurity, but it must be balanced with similar levels of assurance in the software products themselves. Additionally, all of the other commercial and open source applications and capabilities fielded in the DoD must participate in these efforts. This article describes how the federal government and the commercial industries working our nation's critical industries and infrastructure can—by embracing and accelerating the adoption of these standards efforts into all the procurement and development efforts in defense—make the promises of greater assurance and resiliency happen faster and more completely.

will be important in creating needed changes to their operational use and in bringing discipline and repeatability into security measurement [13, 14].

While already making many useful and needed changes possible, these common enumerations and languages will continue to evolve and grow to cover additional areas of standardization. Some of the emerging areas already being worked as standardization efforts are for security events, malware attributes, attack patterns as operational model templates, remediation actions, and interactive survey questions and result reporting (as shown in Table 1, previous page).

These new standardization areas (like those outlined in this article) have the potential to evolve into similarly beneficial standards efforts for those working to secure their enterprises. The key to realizing the promise of these new efforts are the communities working on them and the inclusiveness and fair-handedness of the process surrounding these efforts. The growth and maturity of these new areas should be monitored and evaluated against today's threats and operational challenges. This way, organizations can leverage the ideas and processes as soon as possible and anyone with insights and interests in these areas can get involved.◆

References

1. Martin, Robert A. "The Vulnerabilities of Developing on the Net." CrossTalk Apr. 2001 <www.stsc.hill.af.mil/crosstalk/2001/04/martin.html>.
2. Martin, Robert A. "Transformational Vulnerability Management Through Standards." CrossTalk May 2005 <www.stsc.hill.af.mil/crosstalk/2005/05/0505Martin.html>.
3. Martin, Robert A. "Being Explicit About Security Weaknesses." CrossTalk Mar. 2007 <www.stsc.hill.af.mil/crosstalk/2007/03/0703Martin.html>.
4. Martin, Robert A. "Making Security Measurable and Manageable." CrossTalk Sept. 2009 <www.stsc.hill.af.mil/crosstalk/2009/09/0909Martin.html>.
5. Evans, Karen. "Managing Security Risk By Using Common Security Configurations." *EOP-OMB Memorandum for Chief Information Officers and Chief Acquisition Officers*. 20 Mar. 2007 <www.cio.gov/documents/Windows_Common_Security_Configurations.doc>.
6. Johnson, Clay. "Implementation of Commonly Accepted Security Configurations for Windows Operating Systems." *EOP-OMB Memorandum for Chief Information Officers and Chief Acquisition Officers*. M-07-11. 22 Mar. 2007 <www.whitehouse.gov/omb/assets/omb/memoranda/fy2007/m07-11.pdf>.
7. Evans, Karen S., and Paul A. Denett. "Ensuring New Acquisitions Include Common Security Configurations." *EOP-OMB Memorandum for Chief Information Officers and Chief Acquisition Officers*. M-07018. 1 June 2007 <www.whitehouse.gov/omb/assets/omb/memoranda/fy2007/m07-18.pdf>.
8. Evans, Karen S. "Establishment of Windows XP and VISTA Virtual Machine and Procedures for Adopting the Federal Desktop Core Configurations." *EOP-OMB Memorandum for Chief Information Officers and Chief Acquisition Officers*. 31 July 2007 <www.cio.gov/documents/FDCC_memo.pdf>.
9. Evans, Karen S. "Guidance on the Federal Desktop Core Configuration." *EOP-OMB Memorandum for Chief Information Officers and Chief Acquisition Officers*. M-08-22. 11 Aug. 2008 <www.whitehouse.gov/omb/memoranda/fy2008/m08-22.pdf>.
10. "Twenty Critical Security Controls for Effective Cyber Defense: The Consensus Audit Guidelines." SANS Institute. Vers. 2.3. 13 Nov. 2009 <www.sans. org/critical-security-controls/cag.pdf>.
11. DoD – General Services Administration, and NASA. "Federal Acquisition Regulation; FAR Case 2007–004, Common Security Configurations." 28 Feb. 2008 <http://download.microsoft.com/download/7/6/c/76c0483d-425e-4d99-8d08-15414cf504a2/FAR%202007-004.pdf>.
12. Barrett, Matthew, et al. "Guide to Adopting and Using the Security Content Automation Protocol (Draft)." *NIST SP 800-117*. May 2009 <http://csrc.nist.gov/publications/drafts/800-117/draft-sp800-117.pdf>.
13. "The MITRE Corporation – Information Security Data Standards." *Making Security Measurable*. 10 Sept 2009 <http://makingsecuritymeasurable.mitre.org/list/>.
14. Information Assurance Technology Analysis Center. *Measuring Cyber Security and Information Assurance – State-of-the-Art Report*. 8 May 2009 <http://iac.dtic.mil/iatac/download/cybersecurity.pdf>.

Note

1. The Web sites for these main standardization efforts are: CVE <http://cve.mitre.org>, SCAP <http://scap.nist.gov>, FDCC <http://fdcc.nist.gov>, BSI <http://buildsecurityin.us-cert.gov>, CWE <http://cwe.mitre.org>, CAPEC <http://capec.mitre.org>, CPE <http://cpe.mitre.org>, OVAL <http://oval.mitre.org>, CVSS <www.first.org/cvss>, XCC DF <http://nvd.nist.gov/xccdf.cfm>, CCE <http://cce. mitre.org>.

About the Author

Robert A. Martin is a principal engineer in MITRE's Information and Computing Technologies division. For the past nine years, his efforts have been focused on the interplay of enterprise risk management, cybersecurity standardization, critical infrastructure protection, and the use of software-based technologies and services. Martin is a member of the Association for Computing Machinery, Armed Forces Communications and Electronics Association, IEEE, and the IEEE Computer Society. He has bachelor's and master's degrees in electrical engineering from Rensselaer Polytechnic Institute, and an MBA from Babson College.

The MITRE Corporation
202 Burlington RD
Bedford, MA 01730-1420
Phone: (781) 271-3001
E-mail: ramartin@mitre.org

A DoD-Oriented Introduction to the NDIA's System Assurance Guidebook

Paul Popick
The Aerospace Corporation

Dr. Terence E. Devine
The MITRE Corporation

Rama Moorthy
Hatha Systems

Despite significant strides toward building secure and trustworthy systems, there is ample evidence that adversaries retain their ability to compromise systems. "Engineering for System Assurance" [1] *(the Guidebook)—a publication from the National Defense Industrial Association (NDIA)—provides process and technology guidance to increase the level of systems assurance (SA). One section has guidance particularly useful to the DoD and their contractors. This article provides an introduction to key SA activities with an emphasis on the selected reviews within the DoD Life-Cycle Management Framework.*

For decades, industry and defense organizations have tried to build affordable, secure, and trustworthy systems. Despite significant strides toward this goal, there is ample evidence showing that adversaries retain their ability to compromise systems. Consequently, there is growing awareness that systems must be designed, built, and operated with the expectation that system elements will have both known and unknown vulnerabilities.

SA is defined as:

> The justified confidence that the system functions as intended and is free of exploitable vulnerabilities, either intentionally or unintentionally designed or inserted as part of the system at any time during the life cycle. [1]

This ideal of no exploitable vulnerabilities is usually unachievable in practice, so programs must perform risk management to reduce (to acceptable levels) the probability and impact of vulnerabilities.

This confidence is achieved by SA activities, which include a planned, systematic set of multi-disciplinary activities to achieve the acceptable measures of SA and manage the risk of exploitable vulnerabilities. The *assurance case* is the enabling mechanism showing that the system will meet its prioritized requirements and that it will work as intended in the operational environment, minimizing the risk of exploitation through weaknesses and vulnerabilities.

The Guidebook is intended primarily to aid program managers and systems engineers seeking guidance on how to incorporate assurance measures into their system life cycles. Assurance for security must be integrated into the systems engineering activities to be cost-effective, timely, and consistent. The activities for developing and maintaining the assurance case enable rational decision-making so that only the actions necessary to provide adequate justification (arguments and evidence) are performed. The Guidebook is a synthesis of knowledge gained from existing practices, recommendations, policies, and mandates. SA activities are executed throughout the system life cycle. It is organized based on the ISO/IEC's "Systems and Software Engineering – System Life Cycle Processes" [2]; while there are other life-cycle frameworks, this standard combines a suitably encompassing nature while also providing sufficient specifics to drive SA.

> *"... programs must perform risk management to reduce (to acceptable levels) the probability and impact of vulnerabilities."*

This Guidebook also provides an assurance guidance section for use by the DoD and their contractors and subcontractors. Future editions of this Guidebook may add additional domain-specific assurance guidance.

This article provides an overview of the assurance case section (Section 2.2) and then describes key SA activities completed for selected reviews within the DoD Integrated Defense Acquisition, Technology, and Logistics Life-Cycle Management Framework (referred to in this article simply as the DoD Management Framework) from Section 4 of the Guidebook.

Assurance Case[1]

The purpose of an assurance case is to provide a convincing justification to stakeholders that critical SA requirements are met in the system's expected environment(s). Any assurance claims about the system need to be incorporated into the system requirements.

An assurance case is the set of claims of critical SA properties, arguments that justify the claims (including assumptions and context), and evidence supporting the arguments. The development of the assurance case results in SA requirements that are then flowed to the system architecture and the product baseline. The assurance case can be considered an extension or adaptation of the safety case, which has been used for safety-critical systems. Thus, the concept is not entirely new.

The assurance case need not be a separate document; it may be distributed among or embedded in existing documents. Even if there is an assurance case document, it would typically contain many references to other documents. Regardless of how the assurance case is documented, there must be a way to identify all of the assurance claims, and from those claims trace through to their supporting arguments, and from those arguments to the supporting evidence. For example, an organization might maintain a list of system requirements, tagging specific ones as assurance claims with hyperlinks to arguments that justify why the system will meet the claim. As a minimum:

1. Assurance case claims, arguments, and evidence must be relevant for the system and its operating environment(s).
2. Claims are justified by their arguments.
3. Arguments are supported by their evidence.
4. The assurance case must be developed iteratively, be sustainable, and be maintained throughout the system life cycle

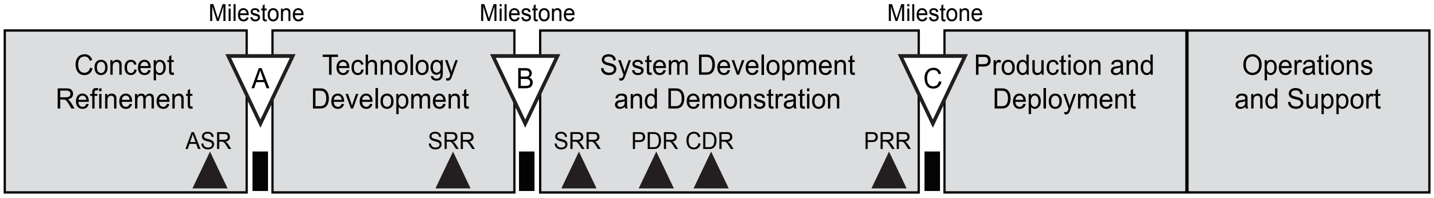


Figure 1: *Life-Cycle Phases with Reviews (from the 2003 Version of DoD Instruction 5000.2)*

as a living document.

5. The assurance case must be delivered as part of the system, to be maintained during system sustainment.

The Guidebook makes no attempt to specify a format for an assurance case, only providing guidance as to what information should be included. The current revision of ISO/IEC 15026 [3] specifies a standard for assurance cases.

The assurance case is generated by the systems engineering technical activities applied to the assurance requirements, and provides evidence of the growing technical maturity of the integration of the SA requirements for use within event-driven technical management. Sections 3 (general) and 4 (DoD-specific) of the Guidebook relate the assurance case to the life-cycle processes.

Figure 2: *ASR Excerpt*

To successfully complete the ASR review, ensure the following SA items were satisfactorily completed:

- System threats and SA claims were considered as part of the analysis of alternatives and full system life-cycle costs were used in the analysis. Sometimes the seemingly cheapest alternative has higher system life-cycle costs due to assurance complications.
- A preliminary identification of critical technologies with a description of how to assure these technologies. This list will eventually feed the Critical Program Information (CPI) developed at the start of the System Development and Demonstration phase. As an aid to this identification, review the Military Critical Technologies List (http://www.dtic.mil/mctl/).
- The Initial Capabilities Document (ICD) and Preliminary System Specification includes:
  - The requirement for the development of an assurance case with high-level claims for each system element determined to be critical.
  - The sustaining mission operational requirements constrain the top-level SA claims to counter identified threats to the mission. They should broadly identify an approach for developing the system assurance case.
  - A critical elements list.
  - The Support and Maintenance Concepts and Technologies with a description of how assurance will be maintained.

Figure 3: *SRR Excerpt*

To successfully complete the SRR, ensure that the following SA items were satisfactorily completed:

- ICD and Preliminary System Performance Specification:
  - Establishes the requirement for the development of an assurance case, including high-level claims for a system determined to be critical.
  - The operational requirements necessary to sustain the mission include the top-level SA claims that address identified threats to the mission that are the foundation for the assurance case. Critical elements are identified.
  - The Support and Maintenance Concepts and Technologies documented with a description of how assurance will be maintained.
  - Requirements with SA implication have been tagged for SA traceability and verification.
- Identification of all critical elements to be protected, and what aspects of them are to be protected (e.g., confidentiality, integrity, availability, authentication, accountability [including non-repudiation], and auditability). For example, ensure that an adversary cannot gain control over a weapon system.
  - Initial identification of potential CPI, and a preliminary approach to the protection of that CPI is part of the system requirements.
  - Identification of all relevant SA threats and their potential impact on critical system assets.

The DoD Management Framework

Section 4 is for use by the DoD and DoD contractors and subcontractors. It is organized according to phases of the DoD Management Framework discussed in DoD Directive 5000.1 [4], DoD Instruction 5000.2 [5][2], and the Defense Acquisition Guidebook (DAG) system life cycle [6].

Section 4's goal is to enable the DoD to acquire or produce assured systems, including both weapons systems and IT systems. This section discusses the topics that should be addressed during the phases of the DoD Management Framework. As discussed in [5] and [6], the life-cycle phases include:

- Concept Refinement.
- Technology Development.
- System Development and Demonstration.
- Production and Deployment.
- Operations and Support.

Section 4.3 is subdivided into DoD review milestones and milestone decision points in this framework. For each review (including the System Engineering Technical Reviews), the DAG description is quoted, followed by a list of the most important SA items to complete prior to that milestone. The non-SA activities normally associated with the review are not specified, but should be considered as the context in which the SA activities are performed. For each review, a specific cross-reference to the corresponding general technical instruction is also provided. Where there are assurance activities that cannot be associated with specific reviews, additional subsections are added to contain them.

Figure 1 depicts the DoD life-cycle phases. The reviews that are discussed in this article are shown in their phases to provide a visualization of when the review occurs. Note that the reviews shown are a subset of life-cycle reviews.

Section 4 focuses on the DoD and makes the assumption that the audience includes system integrators providing systems (both IT and warfighting) to the

DoD environment.

The following subsections provide a description of selected key activities for a subset of the reviews and events.

Alternative Systems Review (ASR)

The ASR occurs during the Concept Refinement Phase prior to Milestone A. Figure 2 shows a partial excerpt of the list of the SA activities to be completed prior to the ASR.

At this point in the life cycle, a key SA activity for each alternative is shown in the first bullet of Figure 2. For each alternative, a threat analysis and the assurance case claims (to counter the identified threats) need to be developed. This analysis needs to be factored into the alternative cost estimates, the selection of a preferred system concept, and the technology development strategy. SA is made more complex when it is not considered in the early stages when alternative concepts are being evaluated.

Systems Requirements Review (SRR)

The SRR can occur at the end of the Technology Development Phase prior to Milestone B, at the start of the System Development and Demonstration Phase, or both. Figure 3 shows a small excerpt of the list of the SA activities to be completed prior to the SRR.

Development of the assurance case claims assists with the development of the system requirements for assurance. Using an assurance case that systematically examines the claims necessary to counter the threats will produce a set of derived security requirements that can be added to the system requirements. Having a robust set of security requirements by SRR allows the security to be built into the system rather than tacked on during the testing or production phase.

Preliminary Design Review (PDR)

The PDR occurs during the System Development and Demonstration Phase, after Milestone B[3]. Figure 4 shows an excerpt from the list of the SA activities to be completed prior to the PDR.

The first bullet of Figure 4 lists the identification of critical components and examination of those components for weaknesses and vulnerabilities. Critical components may be managed through techniques such as graceful degradation, isolation, modularity, diversity, single-point-of-failure reduction/multipathing, and the use of interchange standards to reduce the number, size, and impact of critical elements. Many of these approaches become cost-prohibitive if the weakness and vulnerability analysis is delayed until late-stage testing and production. These activities tie into IA through its control for vulnerability management, known as VIVM-1.

To successfully complete the PDR, ensure the following SA items were satisfactorily completed:

- Use the architecture and preliminary design information (as available) to identify critical components. Identify weaknesses and their associated potential vulnerabilities. Note that a weak architecture can result in a systemic weakness, which can in turn lead to many vulnerabilities. Thus, a rigorous review of the architecture may be required prior to release to design phase. Refine and document a baseline of attack scenarios of the identified threats and assets (see Information Assurance [IA] control: VIVM-1).
- Development of specific instances of SA scenarios, at least for the critical SA requirements, to verify that the system will counter the attack.
- Architecture and preliminary system design includes IA accreditation requirements in its relationship to all hosting enclaves and impact analysis is completed for the architecture. Develop a list of all hosting enclaves as a baseline for tracking purposes as the system moves into the detailed design phase (see IA controls: DCII-1, DCID-1).
    - ◦ Ensure that the architecture and preliminary system design of mobile code usage is evaluated for acceptable risk, avoiding high risk as defined by DoD requirements (see IA control: DCMC-1 [Mobile Code]).
    - ◦ Exclude binary or machine executable public domain software products with no warranty and no source code (see IA control: DCPD-1).

Figure 4: *PDR Excerpt*

Critical Design Review (CDR)

The CDR occurs during the System Development and Demonstration Phase. Figure 5 shows an excerpt from the list of the SA activities to be completed prior to the CDR.

The first bullet in Figure 5 indicates that prior to the CDR the system requirements, the functional baseline, and the allocated baseline need to be updated to incorporate the claims, arguments, scenarios, and any design changes as a result of the assurance case analysis. The fourth main bullet indicates the need to define and select assurance-specific static analysis and criteria for examination during peer reviews (performed during implementation). These are key activities needed to

Figure 5: *CDR Excerpt*

To successfully complete the CDR review, ensure the following SA items were satisfactorily completed:

- Update the system requirements, the functional baselines, and the allocated baseline to incorporate the claims, arguments, and scenarios.
- Capture assurance designs in the associated configuration item build-to documentation as part of the system's Product Baseline. The system's configuration item verification planning should be updated and included in the Product Baseline.
- Update the SA case based on the design, new weaknesses, vulnerabilities identified, and the preceding analysis.
    - ◦ For each SA claim, define the detailed argument(s) to be used to justify the claim, identify the expected evidence (type and expected measure) that will support the argument, and how that evidence will be acquired (including what verification data must be created to acquire that evidence).
    - ◦ After CDR, the assurance case's claims and argument structure are baselined, some evidence is already available, and the methods for acquiring the remaining evidence have been defined.
- Define and select assurance-specific static analysis and assurance-specific criteria to be examined during peer reviews, to be performed during implementation.
    - ◦ Plan for training for assurance-unique static analysis tools and peer reviews.
    - ◦ Ensure that another party (such as a peer) will independently perform static analysis and test, and that the element being reviewed will be the element that will be delivered. This counteracts the risk of a developer intentionally subverting analysis and test, as well as aiding against unintentional errors.

To successfully complete the PRR review, ensure the following SA items were satisfactorily completed:

- Incorporation of the results of system test for weaknesses and their associated vulnerabilities into the assurance case. Verification that the system weaknesses and vulnerabilities have been baselined and appropriately documented (see IA control: VIVM-1).
- Verification that the developed system uses comprehensive test procedures to test any and all patches and upgrades required throughout its life cycle (see IA control: DCCT-1).
- Incorporation of the results of any testing, using industry tools and test cases, for any binary or machine-executable public domain software products with no warranty and no source code being used in the system (see IA control: DCPD-1).
- Evaluation of the relevant test results to obtain the evidence required to build the assurance case from the following list of defensive functions of a system as well as assurance mechanisms that address security, partitioning, access, and traceability mechanisms:
    - ◦ Evaluation of the protection mechanisms with each external interface and associated security requirements using test results as well as other evidence (see IA control: DCFA-1).
    - ◦ Evaluation of the adequacy of security best practices of such functions as identification/authentication (individual and group) using DoD PKI, logon including single sign-on, PKE, key management, smart card, and biometrics (see IA controls: as per DoDI 8500.2, DCBP-1, IATS-2, IAAC-1, IAGA-1, IAIA-2, IAKM-3, ECLO-2, ECPA-1).
    - ◦ Reverification that user interface services are logically or physically separated from data storage and management services. This is particularly important in high assurance systems (see IA control: DCPA-1).

Figure 6: *PRR Excerpt*

ensure the software being developed. Also at this point, the design must meet IA accreditation requirements, including:

- Developing a list of hosting enclaves.
- Evaluating mobile code usage.
- Excluding binary and machine-executable public domain software products with no warranty and no source code.

Production Readiness Review (PRR)

The PRR examines whether the system is ready for production. From an SA standpoint, the test results are examined to ensure that all of the system requirements have been met. This entails looking at test results that substantiate the claims in the assurance case. Figure 6 shows an excerpt from the list of the SA activities to be completed prior to the PRR.

The first bullet discusses the need to incorporate test results for weaknesses into the assurance case as evidence. The weaknesses and vulnerabilities need to be baselined and documented appropriately in accordance with the applicable IA control. The extent to which SA work was performed during the early stages is apparent during the verification of the test results tied to each of the requirements; early emphasis greatly simplifies the verification.

Conclusion

The Guidebook is intended to help alleviate problems by increasing awareness of SA issues, encouraging these issues to be addressed early in the development life

About the Authors

Paul Popick is the associate director of software systems engineering with the Aerospace Corporation. He has more than 35 years of experience in project management and systems engineering management. Popick is a principal author of the NDIA's "Engineering for System Assurance" Guidebook. Prior to joining the Aerospace Corporation, he was director of delivery excellence for a unit of IBM Global Services.

Aerospace Corporation
15049 Conference Center DR
MS CH3-320
Chantilly, VA 20151
Phone: (571) 307-3701
Fax: (571) 307-1833
E-mail: Paul.R.Popick@aero.org

Terence E. Devine, Ph.D., is a software engineer with MITRE. He has more than 35 years of experience in software design and development. Devine is a principal author of the NDIA's "Engineering for System Assurance" Guidebook. He has a doctorate in computer science from UCLA.

The MITRE Corporation
260 Industrial Way West
Eatontown, NJ 07724
Phone: (732) 578-6339
Fax: (732) 578-6014
E-mail: tdevine@mitre.org

Rama Moorthy, CEO of Hatha Systems, has more than 20 years of experience in the high-tech industry, building and delivering products, services, and strategies—both in the commercial and government sectors. In addition to her role as CEO, she supports the DoD's Globalization Task Force on software assurance and supply chain risk management. Moorthy is a principal author of the NDIA's "Engineering for System Assurance" Guidebook. She has a bachelor's degree in electrical engineering and an MBA (marketing and finance).

Hatha Systems
1101 Pennsylvania AVE, NW
STE 600
Washington, D.C. 20004
Phone: (202) 756-2974
E-mail: Rama.Moorthy@
hathasystems.com

cycle, and providing pointers to available resources. The Guidebook shows how SA can be implemented in the existing environment and life cycle. It does not represent original research, but rather a survey of existing work.

The plan is to update the Guidebook to reflect and incorporate feedback received from the programs that use it, December 2008 changes to [5], and new techniques as they emerge. The expectation is that this article will encourage programs to use the Guidebook and to address SA issues early in the life cycle.◆

Acknowledgements

The authors would like to thank the following:

- The NDIA SA Committee Chairs: Kristen Baldwin, Office of the Under Secretary of Defense (Acquisition, Technology, and Logistics); Mitch Komaroff, Office of the Assistant Secretary of Defense (Network and Information Integration); Paul Croll, Computer Sciences Corporation.
- The other principal authors of the Guidebook: David Wheeler, Marie Stanley Collins, Murray Donaldson, and John Miller.
- Arch McKinlay, who wrote the original draft Guidebook's Assurance Case section, which is quoted extensively in this article.

References

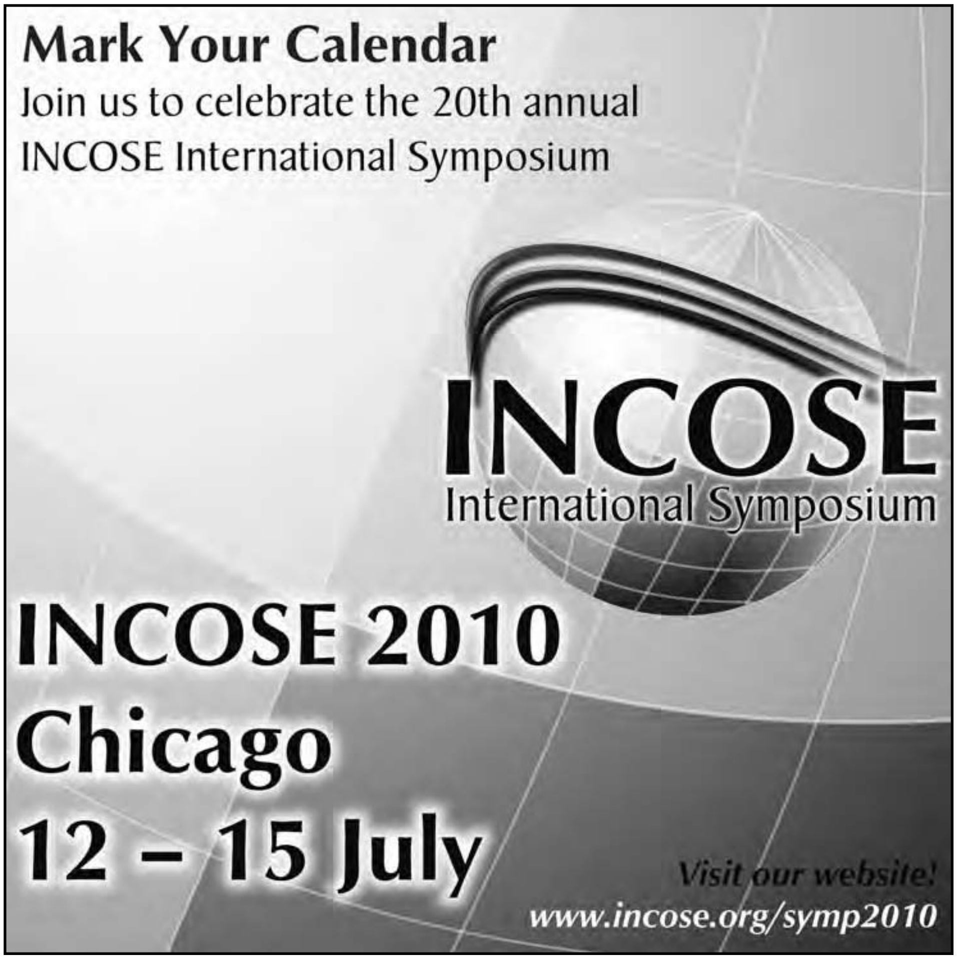
1. NDIA – Systems Assurance Committee. *Engineering for System Assurance*. Oct. 2008 <www.acq.osd.mil/sse/docs/SA-Guidebook-v1-Oct2008.pdf>.
2. ISO/IEC. *Systems and Software Engineering – System Life Cycle Processes*. International Standard 15288-2008. 1 Feb. 2008.
3. ISO/IEC. *Systems and Software Engineering – Systems and Software Assurance*. International Standard 15026. Draft.
4. DoD. *The Defense Acquisition System*. Directive 5000.1. 2003.
5. DoD. *Operation of the Defense Acquisition System*. Directive 5000.2. 12 May 2003.
6. Defense Acquisition University. *Defense Acquisition Guidebook*. 12 Dec. 2004.

Notes

1. The Introduction and Assurance Case sections of this article are excerpted from [1].
2. The Guidebook is written the 2003 release of 5000.2 [5]; although the current release (2008) has changed the phases, the Guidebook 5A instructions for the systems engineering technical review criteria remain applicable.
3. The 2008 release of DoD Instruction 5000.2 now has the PDR prior to Milestone B.

Additional Resources

1. DoD. *Information Assurance*. Directive 8500.01E. 23 Apr. 2007 <www.dtic.mil/whs/directives/corres/pdf/850001p.pdf>.
2. DoD. *Acquisition Systems Protection Program*. Directive 5200.1-M. Mar. 1994 <www.dtic.mil/whs/directives/corres/pdf/520001m.pdf>.
3. DoD. *National Industrial Security Program*. Directive 5220.22. 1 Dec. 2006 <www.dtic.mil/whs/directives/corres/pdf/522022p.pdf>.
4. DoD. *Defense Intelligence Agency*. Directive 5105.21. 18 Mar. 2008 <www.dtic.mil/whs/directives/corres/pdf/510521p.pdf>.
5. DoD. *Security, Intelligence, and Counterintelligence Support to Acquisition Program Protection*. Directive 5200.39. 10 Sept. 1997 <http://fas.org/irp/doddir/dod/d5200_39.pdf>.
6. DoD. *Operation of the Defense Acquisition System*. Instruction 5000.2. 8 Dec. 2008 <www.dtic.mil/whs/directives/corres/pdf/500002p.pdf>.
7. DoD. *Information Assurance Implementation*. Instruction 8500.2 <www.dtic.mil/whs/directives/corres/pdf/850002p.pdf>.
8. DoD. *DoD Information Assurance Certification and Accreditation Process*. Instruction 8510.01. 28 Nov. 2007 <www.dtic.mil/whs/directives/corres/pdf/851001p.pdf>.
9. Information Assurance Technology Analysis Center. *DoD Information Assurance and Computer Network Defense Strategies: A Comprehensive Review of Common Needs and Capability Gaps – State-of-the-Art-Report*. 21 July 2005 <http://iac.dtic.mil/iatac/pdf/gap.pdf>.
10. DHS. *Security in the Software Lifecycle: Making Software Development Processes—and the Software Produced by Them—More Secure*. Draft 1.2. Aug. 2006 <www.sis.uncc.edu/~seoklee/teaching/Papers/SwA%20Security%20in%20the%20Software%20Lifcycle%20v1.2%20-%20091306.pdf>.
11. Redwine, Samuel T., Jr., Ed. *Software Assurance: A Curriculum Guide to the Common Body of Knowledge to Produce, Acquire, and Sustain Secure Software*. Version 1.2. U.S. Department of Homeland Security. Oct. 2007 <https://buildsecurityin.us-cert.gov/daisy/bsi/resources/dhs/927-BSI.html>.



Meaningful and Flexible Survivability Assessments: Approach and Practice©

Michael Atighetchi and Dr. Joseph Loyall
Raytheon BBN Technologies

With the increase of IT assets and attacks against them—along with increased attention to, and concern for, cybersecurity and the survivability of systems—systems assurance assessment has become more important than ever. This article describes a flexible process for assessing systems with respect to security and survivability. We discuss how our approach enables assessments throughout the project life cycle by tailoring the testing to a specific evaluation scope in terms of depth and coverage.

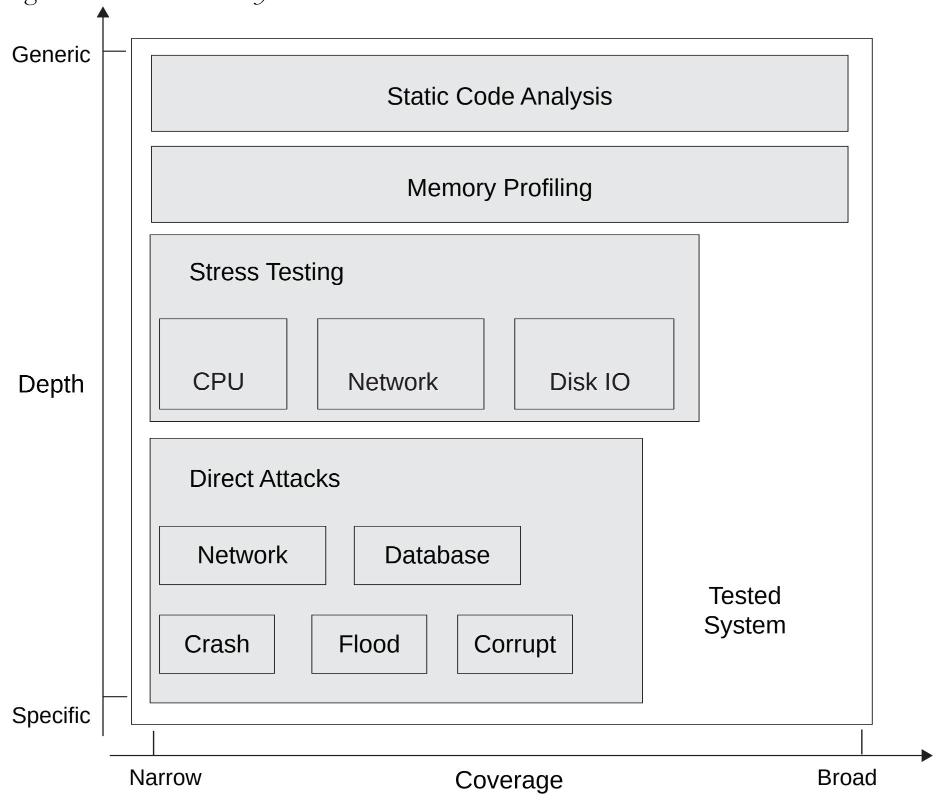
Current systems assurance testing tends to focus on functional and performance testing and often postpones security assessments until the end of the project life cycle—or after deployment or release. In our view, this shortcoming can be partially explained by the large cost and technical difficulty of formal security testing (e.g., during certification and accreditation), and the lack of standardized non-binary security metrics. For example, the testing guide of the Open Web Application Security Project (OWASP) [1] contains more than 340 pages on good security testing procedures but provides little guidance in selecting which tests to perform for a given application.

In this article, we present a flexible process for assessing evolving prototype systems with respect to security and survivability. Also discussed is how our approach enables assessments of confidentiality, integrity, and availability (CIA, a concept detailed in the sidebar) throughout the project life cycle by tailoring the testing to a specific evaluation scope in terms of depth and coverage. This process has enabled us to conduct security evaluations early in project life cycles and focus architecture and design efforts on addressing the shortcomings identified through repeated tests.

To show the testing methodology in action, this article provides details on a set of specific open-source and freely available tools we have found useful in our evaluations. The focus of this article is not on comparing different tools and their capabilities, but rather on showing, through insight from our *test attacks*, what vulnerabilities can be identified and how these tools can be reused across evaluations of different systems.

Finally, we report on the most recent assessment of an information management system (IMS), implementing a publish, subscribe, and query (PSQ) capability.

Figure 1: *The Survivability Evaluation Process*



After giving a brief testing methodology overview, this article will detail static code analysis and memory profiling tools used in capturing a broad set of coding mistakes, present significant hands-on technical knowledge that can be used to study risk exposure CIA loss, and propose the next steps in both improving survivability assessments and mitigating vulnerabilities.

Survivability Assessment Methodology

Continuous test and evaluation of security attributes of systems is an important part of testing as it allows software developers to identify and address vulnerabilities as part of the system architecture and design. This article describes a repeatable survivability testing methodology that optimizes the set of tests to execute given limited budget constraints and can run as part of continuous build processes. Figure 1 displays the main threads of our testing methodology as boxes.

First, we utilized existing tools for *static code analysis* and *memory profiling*. These tools provide excellent coverage at minimal cost but only support validation against a set of specialized known problems. Next, we performed a series of stress tests that simulated conditions found in operational environments, which tend to be less controllable than lab environments. Tests in this category focused on studying the impact of a large number of clients and large size information objects on critical server functionality. These tests were narrower in scope than static code checks and memory profiling, symbolized by the "Stress Testing" box in Figure 1. Finally, we developed a set of direct attacks against the system with different attacker privileges, ranging from attacks launched with only network layer access to attacks that assumed corrupted clients. These attacks focused on exploiting specific vulnerabilities, therefore providing the least amount of coverage but the most amount of depth. Figure 1 visualizes our approach of subjecting the whole system to a set of low-cost generic checks to maximize coverage while performing a more in-depth analysis on carefully selected parts of the system. As the figure indicates, the methodology enables a flexible amount of testing from *specific* (i.e., to the requirements of the tested program) to *generic* (i.e., testing that could be applied to any program) and from *narrow* to *broad* coverage of the tested system. The depth and coverage of the testing can be chosen to enable the best testing possible in given time and budget constraints.

The CIA Triad

Confidentiality (C) is the assurance that information is not disclosed to unauthorized individuals, programs, or processes. For example, a credit card transaction on the Internet requires the credit card number to be transmitted from the buyer to the merchant and from the merchant to a transaction processing network. The system attempts to enforce confidentiality by encrypting the card number during transmission, by limiting the places where it might appear (e.g., in databases, backups, or printed receipts), and by restricting access to the places where it is stored. If an unauthorized party obtains the card number in any way, a breach of confidentiality has occurred.
Integrity (I) is the assurance that information is not altered in an unauthorized fashion. For example, integrity is violated when an employee deletes important data files, when a computer virus infects a computer, or when an employee is able to modify his own salary in a payroll database.
Availability (A) is the assurance that information, systems, and resources are available to users in a timely manner so productivity will not be affected.

Static Code Analysis and Memory Profiling

The purpose of static code analysis is to automatically flag common coding errors that leave the system vulnerable to exploits. A large number of code analysis algorithms and tools for finding security vulnerabilities exist, including both commercial- and research-grade tools, with different trade-offs for cost and effectiveness.

We utilized several analysis tools as part of our research efforts (as described in the following text). We did not conduct a comprehensive comparison of the available analysis tools, but instead selected a representative set that provided useful analysis. There are alternatives to the particular tools that we selected, although the amount of coverage and analysis differs between tools. Furthermore, we concentrated on tools that analyze programs written in Java and C++. Arguably, these are the most common languages in use today and are the languages in which the systems that we tested were written. We believe the concepts and process that we put forward are valid for any language (although the set of available tools will certainly vary and the tools for languages other than Java and C++ might be less readily available).

FindBugs [2] was the most useful tool for us when analyzing Java code. FindBugs uses static analysis to inspect Java byte code for occurrences of bug patterns without the need to execute the program. For analyzing C++ code, we used FlawFinder [3] and RATS [4] open-source tools. FlawFinder is a program that examines source code and reports possible security weaknesses (*flaws*) sorted by risk level. It is useful for quickly finding and removing some potential security problems before a program is released. RATS is a tool for scanning C, C++, Perl, PHP, and Python source code, and for flagging common security-related programming errors such as buffer overflows and time-of-check-to-time-of-use race conditions.

In order to detect memory leaks (which can adversely affect availability) and memory corruption (which can be exploited to escalate privilege through buffer overflow attacks), this phase of the testing process runs a system through a set of memory profiling tools that keep track of dynamic memory allocations and perform analysis to detect errors during operation.

For C++ programs, we successfully used Valgrind [5], Mudflap [6], and mpatrol [7]. This set was selected to maximize coverage and mitigate errors introduced by individual tools. For Java programs, the scope of memory profiling is not to directly detect bad memory conditions (since Java avoids memory corruption and leak issues through garbage collection), but to provide useful debugging functionality in cases where Java runs out of memory.

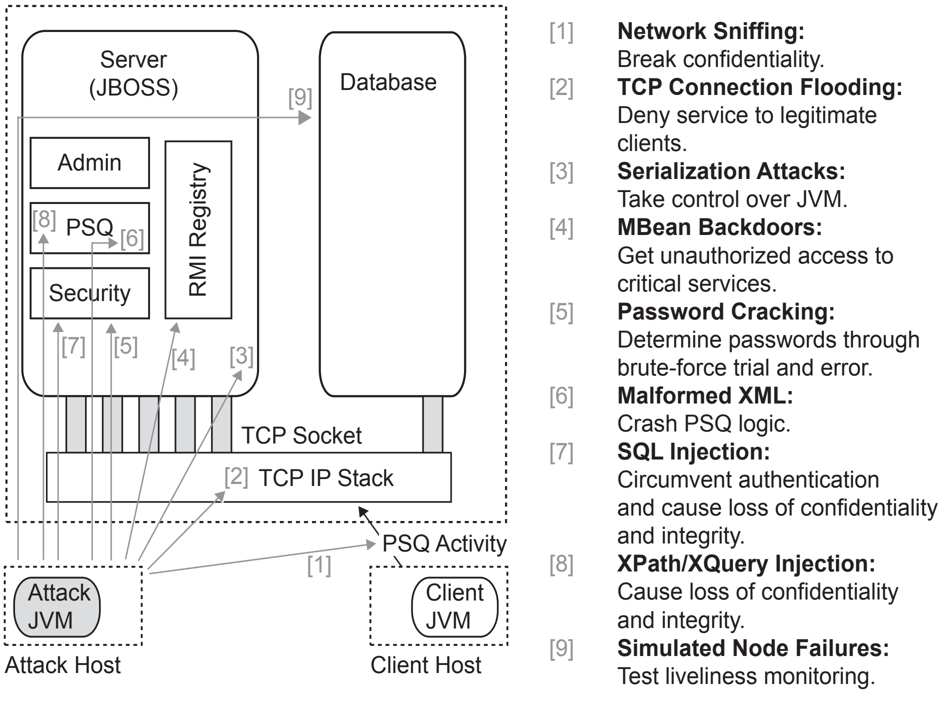
Stress Testing

The goal of stress testing is to assess system functionality under application-level boundary conditions, such as high load scenarios with large information objects, large numbers of clients, and high rates of client requests.

Large MIOs

During the IMS assessment, we studied whether a single client can affect the availability of the IMS server by publishing a few very large managed information

Figure 2: *Progression of Direct Attacks that Compromise CIA*

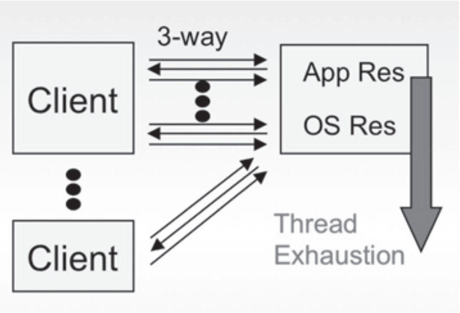

Server
(JBOSS)
Database
[9]
Admin
[8]
PSQ
[6]
RMI Registry
Security
[7]
[5]
[4]
[3]
TCP Socket
[2] TCP IP Stack
PSQ Activity
[1]
Attack
JVM
Client
JVM
Attack Host
Client Host
[1] Network Sniffing: Break confidentiality.
[2] TCP Connection Flooding: Deny service to legitimate clients.
[3] Serialization Attacks: Take control over JVM.
[4] MBean Backdoors: Get unauthorized access to critical services.
[5] Password Cracking: Determine passwords through brute-force trial and error.
[6] Malformed XML: Crash PSQ logic.
[7] SQL Injection: Circumvent authentication and cause loss of confidentiality and integrity.
[8] XPath/XQuery Injection: Cause loss of confidentiality and integrity.
[9] Simulated Node Failures: Test liveliness monitoring.

objects (MIOs)[1]. To study the impact of MIO size, we modified an existing test client to publish MIOs with various metadata and payload sizes. The goal of this test was to identify the boundary point at which MIOs get rejected by the server due to their size. For scenarios with no active subscriptions, a client publishing an MIO with a combined payload size of 28MB (payload = 14MB and metadata = 14MB) caused exceptions in the core IMS and sometimes on the client side, leading to loss of the MIO. This was well below the maximum Java Virtual Machine (JVM) memory limits of 1024MB on both the client and server side and represented an inadvertent internal size limit that could be exploited by rogue clients. In a follow-on experiment, we studied the impact of the number of active subscriptions on the maximum accepted MIO size. One active subscription reduced the maximum MIO size to 5.8MB, while only MIOs with less than 2MB could be delivered to two subscribers. The expected cause was prolific copy operations on the metadata strings during subscription predicate matching. The IMS code has been tested with large payloads but never with MIOs with large metadata based on an undocumented assumption that metadata is usually small compared to payloads. While this assumption may hold in practice during normal use, it is the type of assumption likely to be exploited by a determined adversary. The solution to this problem is to refactor the code to avoid duplication of byte arrays.

Large Number of Clients

Another stress test focused on evaluating the impact of a large number of concurrent clients on IMS server availability. For that purpose, we simply started multiple publishing clients on the same client host. Out of memory exceptions occurred in the core IMS after registering 36 clients, resulting in a denial of service to all clients. In addition to identifying a point of optimization to increase the number of clients supported, this test also pointed out a survivability and security need. Since there is always going to be an upper limit to the number of clients that can be properly supported by a single server, there should be code in the server that checks the number of clients and refuses new clients when the limit is reached.

Figure 3: *TCP Connection Flooding Attack*



Direct Attacks

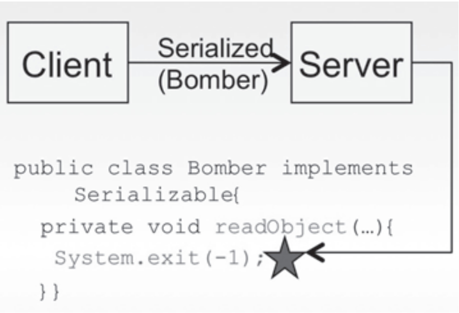
Adversaries typically don't start from scratch when attacking applications and can leverage a large collection of openly available tools to construct customized multi-step attacks targeting critical application functionality. Figure 2 (see previous page) displays a progression of attacks (with an increasing level of privilege) that an adversary might follow to compromise CIA. This attack sequence crosses multiple system and protection boundaries, as is common during sophisticated attacks. The following subsections describe each attack in more detail, describe observations made during test attacks, and provide suggestions on mitigation strategies.

> *"Adversaries ... can leverage a large collection of openly available tools to construct customized multi-step attacks targeting critical application functionality."*

Network Sniffing

Description: Attackers use sniffing to find out information about the system (for attack purposes) and to steal sensitive information.
Risk: Loss of confidentiality and integrity.

Figure 4: *Java Serialization Attack*



```
public class Bomber implements
    Serializable{
  private void readObject(…){
   System.exit(-1);
  }}
```

Example: Wireshark [8] was used to capture the raw content of packets sent out by a publishing client to the IMS core. Although the IMS uses Transport Layer Security (TLS) [9] for encrypting data on the network and preventing sniffing attacks, it was noticed that TLS was only used for initial authentication and connection establishment between clients and the server. Published MIOs went over a separate Transmission Control Protocol (TCP) connection that did not provide TLS protection. As a result, the IMS was vulnerable to a loss of confidentiality via standard network sniffing. Furthermore, attackers could cause integrity violations by exchanging MIO payloads (e.g., swapping out targets in a target nomination list). Traffic was also vulnerable to replay attacks, which are a special form of data corruption where the data content isn't changed but corruption is still introduced by republishing MIOs.
Mitigation: The solution to these vulnerabilities is protecting all communication between clients and the server via TLS, and devising automated tests that classify all observed traffic.

TCP Connection Flooding

Description: The main idea of the TCP connection flood is to initiate and establish a large number of TCP connections from an infiltrated client to a listening socket. Since servers typically set aside memory and process resources for new connections, a large number of connections can create memory exhaustion in which further connections from legitimate clients will be dropped. The top of Figure 3 shows the normal *three-way handshake* during TCP connection establishment. A client initiates the connection via a synchronize (SYN) packet, which gets answered by the server with a SYN/Acknowledge (ACK) packet. Upon receiving the SYN/ACK packet, the client replies with another ACK packet, which establishes the TCP connection.

TCP connection floods are different from SYN floods, in which the client simply goes on to send out a large number of SYN packets originating either from the same client IP address or multiple (*spoofed*) client IP addresses without completing the three-way handshake. Each SYN packet needs to get processed by the server's TCP IP stack; such a situation can cause memory and CPU exhaustion in the server's TCP IP stack. Modern operating systems deal with SYN floods effectively using SYN cookies [10].

Figure 3 displays the sequence of events during a TCP connection flooding attack. Here the client completes the three-way TCP handshake, but then keeps creating new connections. A straightforward version of this attack keeps the client's source IP of all connections the same, whereas more sophisticated versions can spread to multiple client IPs. Note, however, that this escalation cannot simply be achieved through IP address spoofing, since the client needs to get the server's SYN/ACK packet routed to it in order to proceed. Since each established connection generally causes application resources to be used (e.g., threads and associated memory structures), a large number of connections can cause applications to run out of memory, causing legitimate clients to fail.

Risk: Loss of availability.

Example: In an experiment to assess the IMS software's vulnerabilities to TCP connection floods, we used an attack client that established and held 1,200 TCP connections, and studied the effect of pointing this client to each of 15 listening ports in separate runs. The results were interesting. The maximum number of established connections varied across ports (from 60 to 753) as did attack effects. This included:

- A loss of publish functionality.
- A loss of administration functionality.
- A loss of system access on the node running the IMS server process due to maximum file handle limits.
- Denial-of-service through disk exhausting via large log files.
- No log generation for floods on certain ports, enabling attackers to execute stealthy attacks that are hard to diagnose.

Mitigation: A threshold scheme taking into account the source IP of the connection together with past connection history makes execution of this attack significantly harder. In addition, code to limit the rate of outgoing TCP connections from legitimate clients helps prevent such conditions in the case of accidental application programming interface misuse in client programs. Furthermore, ports that only require local access (e.g., 5400) should be bound to 127.0.0.1 instead of 0.0.0.0 to prevent remote execution of connection flooding attacks. Finally, all code should be augmented with proper exception handling to generate a small number of succinct exceptions in flooding conditions.

Serialization Attacks

Description: This attack exploits known type safety issues Java programs encounter during deserialization. Creating and sending a special serialized Java object to a remote method invocation (RMI) server port (as shown in Figure 4) allows attackers to remotely exploit JVM bugs and cause client-initiated execution of arbitrary code.

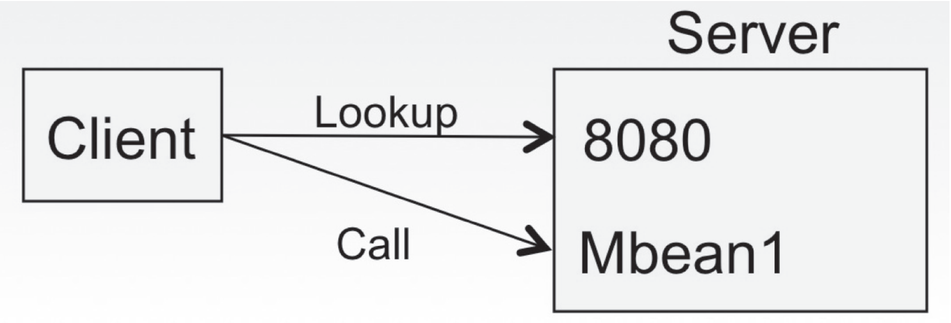


Figure 5: *Direct Calls on MBeans Without Authentication*

Risk: Loss of CIA.

Example: Let's consider the following server code snippet used for reading objects off the network, as described in [11]:

```
  mySocket = new ServerSocket(3000);
  while (true) {
    Socket client = mySocket.accept();
    ReceiveRequest dtwt = new Receive
Request (client);
}
class Request implements Serializable { }
class ReceiveRequest extends Thread{
    Socket clientSocket = null ;
    ObjectInputStream ois = null;
    ReceiveRequest (Socket theClient)
throws Exception {
     clientSocket = theClient;
     // get the Streams
     ois = new
          ObjectInputStream(clientSocket.
get InputStream());
}

   public void run() {
    try {
      Request ac = (Request)
                          t=2
                     ois.readObject(); }
                          t=1
      catch (Exception e)
{  System.out.println(e) ; }
      // ...
    }
 }
```

Note how the server first reads in the Object (t=1) and then performs the cast to the expected Request type (t=2). The generated byte code shows the following execution sequence:

```
public void run();
     Code:
     0:   aload_0
t=1   1:  getfield          #3; //Field
                            ois:Ljava/io/Object
                            InputStream;
      4:  invokevirtual  #7; //Method
                     java/io/ObjectInputStream.
                     readObject:()
                     Ljava/lang/Object;
t=2   7:  checkcast      #8; //class Request
     10:  astore_1
     11:  goto     22
     14:  astore_1
     …
```

The sequence of events is as follows:

- t=0 client sends byte stream (serialized object data) via ObjectInputStream.
- t=1 Server branches into the readObject method of the class according to the result returned by getfield.
- t=2 server casts the object to the needed type.
    - o Cast is valid: continue work.
    - o Cast is invalid: throw ClassCast Exception.

Between t=0 and t=2, there is no type safety. The client gets to decide (at t=0) which code the server branches into (at t=1). This opens up an attack path in conjunction with an existing vulnerability of a readObject method on any class that is on the server's load path. First, attackers can find some vulnerable class definitions on the server (especially in a readObject method, any serializable class will do). Next, they can construct an object according to this class definition and finally embed the malicious object in the ObjectInputStream payload of the Java 2 Platform Enterprise Edition (J2EE) protocol (RMI, RMI/Internet Inter-Orb Protocol, Java Naming and Directory Interface [JNDI], and so forth). Serialization attacks have been used frequently in the past: see [11] for regular expression exploits and [12] for exploiting hash table collisions.

Mitigation: Our suggestions to prevent serialization vulnerabilities involve careful review and minimization of loadable classes on the server's classpath. In addition, a crumple zone—a layer of proxy components that anyone seeking service must go through—can be deployed to perform the deserialization of all objects entering the core. Augmenting proxies with proper monitoring and restart capabilities, and introducing diversity through different JVM implementations and programming languages, will help reduce the dangers of this attack type.

MBean Backdoors

Description: This attack attaches a remote client to the managed beans (MBeans) available—via Java Management Extensions inside of a J2EE server—and makes dynamic calls into the server without the need for authentication. As shown in Figure 5 (see previous page), clients can make direct calls on MBeans handled by a JBoss Application Server (AS) by simply performing a lookup, creating an RMIAdaptor connection, and making RMI calls.

Risk: Loss of CIA.

Example: In the IMS code, clients could directly connect to the MBeans via jmx/invoker/RMIAdaptor and make calls through the MBeanServerConnection without needing to authenticate. This would enable calls that could remove all repositories, inject a man-in-the-middle master repository (loss of confidentiality), and change security policies (loss of integrity), for example. No logging is performed during these remote calls, which makes this attack very stealthy.

Mitigation: To prevent this attack, one needs to harden access to 127.0.0.1:8080/invoker/JNDIFactory and lookup of jvm/invoker/RMIAdaptor. Using TLS and binding, the listening socket to 127.0.0.1 only will make access by unauthorized clients more difficult. In addition, tighter integration of the socket listener on port 8080 with security handling code helps prevent this attack.

Password Cracking

Description: This attack guesses correct passwords through repetitive trials of passwords retrieved from common attack dictionaries. Once the valid administrator password has been determined, the adversary can simply log in as the administrator and perform all actions normally granted to administrators, including deletion of critical MIOs and removal of users.

Risk: Loss of CIA.

Example: The prototype IMS code was susceptible to brute force password attacks because it allowed attack scripts to try an unlimited number of password combinations without impacting account status. No log events occurred in the JBoss AS console when authentication failed, making the attack difficult to detect.

Mitigation: An effective approach to counter-password attacks is to establish a cutoff scheme in which accounts are locked down after a specific number of failed login attempts. In addition, failed login attempts should be logged to a management station.

Malformed XML

Description: These attacks attempt to deny service to legitimate clients by taking over a single client and publishing malformed XML input data with the intent to crash the server.

Risk: Loss of availability.

Example: We tested the vulnerability of the prototype IMS to this kind of attack by modifying the client to publish MIOs with improperly formatted metadata. XML validation properly caught and handled the bad content. Next, we tried uploading a malformed schema and observed the following issues:

- Schemas were properly validated, but validation exceptions cause ClassNotFoundExceptions.
- Schemas that don't start with <?xml version passed validation but caused errors during publication.
- Schemas with duplicate xsd:schema lines passed validation but cause Null Pointer Exceptions in the Berkeley DB XML backend, and cause Web console access to the Interoperable Object Reference to fail.

These issues are easy to fix but are indicative of the bugs that can creep in, even when XML content is validated through schemas but the schema itself is not.

Mitigation: The use of restrictive XML schemas and validation of all XML input against them goes a long way in addressing this vulnerability.

Figure 6: *Data Propagation to SQL Databases (SQL DBs)*

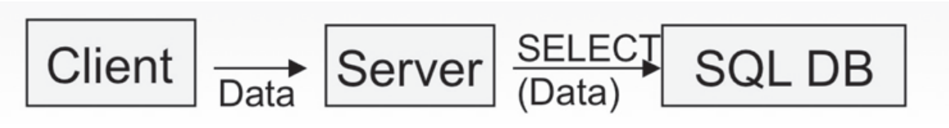
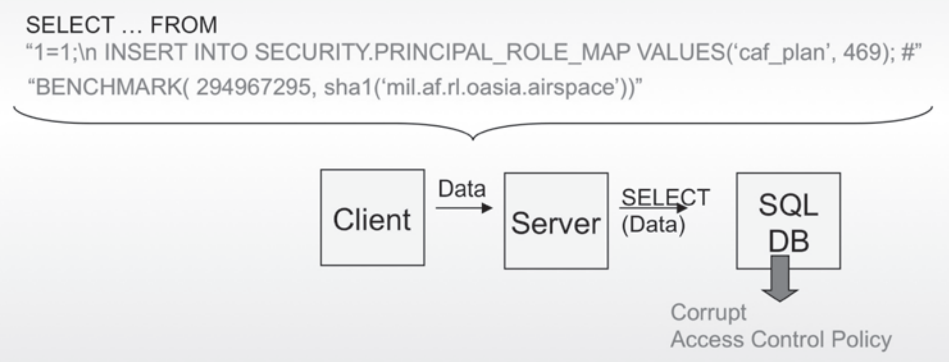


Figure 7: *Example SQL Injection Attack*



```
SELECT … FROM
"1=1;\n INSERT INTO SECURITY.PRINCIPAL_ROLE_MAP VALUES('caf_plan', 469); #"
"BENCHMARK( 294967295, sha1('mil.af.rl.oasia.airspace'))"
```

SQL Injection

Description: SQL injection is a technique that exploits a security vulnerability occurring in the database layer of an application. The vulnerability is present when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and, thereby, unexpectedly executed. It is (in fact) an instance of a more general class of vulnerabilities that can occur whenever one programming or scripting language is embedded inside another.

Risk: Loss of CIA.

Example: Figure 6 displays the normal flow of data from a client via a server to the backend SQL DB. If the data sent by the client is not filtered before reaching the SQL DB, the client can execute arbitrary commands on it.

Figure 7 depicts one such example that was used by attackers during the 2005 OASIS Dem/Val red team exercise [13]. The attack locked up servers by inserting a BENCHMARK command, which would cause the SQL DB to use 100 percent of the available CPUs. The BENCHMARK command causes MySQL (a relational database management system) to run through an extensive set of mathematically expensive computations, pre-empting useful computation for hours.

Our testing indicated that the prototype IMS code was not vulnerable to SQL injection attacks.

Mitigation: To prevent SQL injection attacks, user interfaces should be designed to be as restrictive as possible (e.g., a selection menu instead of free text entry, and all user input should be checked for embedded SQL commands).

XPath/XQuery Injection

Description: This attack attempts to gain the same effects described in the SQL injection attack by providing specially crafted query predicates.
Risk: Loss of CIA.
Example: The following code fragment shows an example in which the client explicitly specifies parts of the XQuery that usually gets implicitly generated and executed by the IMS server:

```
connection.createQuerySequence
            ("mil.af.rl.oim.training.xmlx-
path", "1.0");
String xmlxpathPredicate =
Utils.createPredicate
  ("TestPredicate", "XQuery",
   "for $a in
collection(\"SCHEMA_422547673.dbxml\")
    /child::sys:Metadata
    where ($a//ATO/type = \"ATO\")

returndbxml:metadata(\"dbxml:name\",
$a)");
```

XPath injection attacks proved partially effective against the prototype IMS, partly because it dynamically constructed and executed XQuery statements from XPath input. Custom XQuery predicates allow a small amount of information exfiltration. For instance, an attack client could probe through repeated queries to determine whether air task orders (ATOs) were available with certain metadata fields, although the client could not actually retrieve the ATO payload. In addition, the IMS, which explicitly prohibited full XQuery predicates because of their ability to modify the database, included a *backdoor* through which full XQuery commands could be sent to the database. This only presented a vulnerability if the backdoor code persisted through releases of the code. In addition to vulnerabilities to the IMS database, XQuery is a powerful language, allowing (in principle) direct calls into the JVM through external functions. This would allow an attacker, for instance, to call `System.exit(-1)` through a specially crafted query. The prototype IMS did not exhibit this vulnerability because it used an earlier version of XQuery that did not support embedded Java functions.
Mitigation: To prevent XQuery exploits, all user input should be checked for dangerous XQuery commands, such as `delete`, `update`, and references to external functions.

Software Defense Application

This article describes survivability assessment techniques that will benefit the defense software community through more flexible design evaluations, more useful assessments, and reduced time in identifying and mitigating vulnerabilities. The in-depth analysis of the most prevalent cyberattack scenarios will help defense software developers understand what they may encounter; but, more importantly, the authors present first-hand accounts detailing how they solved these problems. The end result is a significant increase in a system's overall security.

Simulated Node Failures

Description: A common survivability experiment involves studying the impact of crashes of one component on other components. However, it can be difficult to devise an attack that is so precise that it crashes only the targeted component. Therefore, this attack uses fault injection techniques to simply cause the desired fault (e.g., by manually stopping a component).
Risk: Sustained loss of availability.
Example: We injected crash faults in the IMS database (Berkeley DB XML) by sending the database a `kill -9` signal. The IMS did not contain a monitoring protocol to test the liveliness of processes. An accidental crash of the Berkeley DB XML process led to situations in which the IMS server was unavailable, and the situation was noticed only when critical operations started failing.
Mitigation: A secure, heartbeat-based monitor that provides low failure detection latencies and is resistant to spoofing attacks. The monitor can reduce the impact and speed the recovery from these attacks.

Results and Summary

Table 1 displays a summary of both stress tests and direct attacks conducted, their results, and their effects on CIA.

In summary, nine of the 11 tests were successful in achieving attack objectives. Risks to confidentiality were exposed in five tests, which is a surprisingly high number given that it is generally much easier to cause a denial of service than to exfiltrate data without being detected. The experiment generated new requirements for the IMS—both in terms of fixing bugs and ease-of-use issues as well as in addressing deeper problems, including a single point of failure for various components, lack of adequate management tools, and susceptibility to direct attacks

This article presented a flexible process for evaluating systems early on in their life cycles for information survivability and showed its application during an assessment of a prototype PSQ information management system. This work

Table 1: *IMS Assessment Results*

Test	Result	Effect*
Large MIOs	Out of memory errors with IOs over md = 14MB, pl = 14MB, 0 subscribers md = 5.8MB, pl = 5.8MB, 1 subscriber md = 2MB, pl = 2MB, 2 subscribers.	A
Large Number of Clients	Out of memory errors after registration of 36 clients.	A
Network Sniffing	MIOs are sent in the clear.	CI
TCP Connection Flooding	Denial-of-service without generating log entries.	A
Serialization Attacks	Clients can execute arbitrary code in the JVM without authentication.	CIA
MBean Backdoors	Clients can make anonymous calls on MBeans.	CIA
Password Cracking	Brute-force password testing at the rate of 100 per second.	CI
Malformed User Data	Apollo resilient against malformed metadata and schemas.	None
SQL Injection	Suspicious SQL code turns out to be dead code.	None
XQuery Injection	Unauthorized access to MIOs and execution of arbitrary code.	CIA
Simulated Node Failures	Absence of monitoring protocol.	A

*C = Confidentiality, I = Integrity, A = Availability   pl = Payload, md = Metadata builds upon previous and ongoing research in survivable distributed systems and addresses the current and future need to effectively and accurately evaluate system guarantees in the presence of sophisticated cyberattacks. We have packaged a number of reusable attack techniques and associated tools that can be customized for new systems with little overhead.

Next Steps

Our future research will continue to build upon the work presented in this article in two ways.

First, we plan to extend our assessment approach by adding more techniques and automating customization and automation aspects of attack scenarios. This will further reduce overhead costs for assessments and increase their frequency during a project life cycle. We clearly see the value of establishing a community-based collaboration platform for survivability assessments (e.g., those hosted on <https://www.forge.mil>). As more projects start using the techniques described in this article, system survivability requirements will need to be taken into account as systems need to provide different levels of survivability. More research is needed to develop a point/scoring system that evaluates survivability in a systematic and quantitative way—given the security posture and criticality of an application.

Second, we plan on investigating solutions to the deeper problems of the existence of single points of failures and the lack of adequate management tools. For future work in service-oriented information management systems, we intend to:

1. Extend service-oriented architecture and design techniques to include security and survivability concepts that facilitate survivable designs.
2. Develop security services, mechanisms, and execution containers for preserving system CIA in the presence of cyberattacks.
3. Develop an environment to assess and evaluate composition patterns, enabling customization of tradeoffs between survivability, performance, and functionality for specific environments.◆

Acknowledgements

The authors would like to acknowledge the support and collaboration of the U.S. Air Force Research Laboratory (AFRL) Information Directorate and the ITT Corporation. This work was sponsored by the AFRL under contract number SPO700-98-D-4000, Subcontract Number: 205344 – Modification No. 5.

About the Authors

Michael Atighetchi is a scientist at Raytheon BBN's Information and Knowledge Technologies business unit. His research interests include cross-domain information sharing, security and survivability architectures, and middleware technologies. Atighetchi has published more than 35 technical papers in peer-reviewed journals and for conferences, and is a senior member of the IEEE. He holds a master's degree in computer science from the University of Massachusetts at Amherst, and a master's degree in IT from the University of Stuttgart, Germany.

Raytheon BBN Technologies
10 Moulton ST
Cambridge, MA 02138
Phone: (617) 873-1679
E-mail: matighet@bbn.com

Joseph Loyall, Ph.D., is a principal scientist at Raytheon BBN Technologies. He was the principal investigator for Defense Advanced Research Projects Agency and AFRL research and development projects in the areas of information management, distributed middleware, adaptive applications, and quality of service. He is the author of more than 75 published papers; was the program committee co-chair for the Distributed Objects and Applications conference (2002, 2005); and has been invited speaker at several conferences and workshops. Loyall has a doctorate in computer science from the University of Illinois.

Raytheon BBN Technologies
10 Moulton ST
Cambridge, MA 02138
Phone: (617) 873-4679
E-mail: jloyall@bbn.com

References

1. OWASP Foundation. *OWASP Testing Guide*. Vers. 3.0. 2008 <www.owasp.org/images/5/56/OWASP_Testing_Guide_v3.pdf>.
2. The University of Maryland. "Find Bugs – Find Bugs in Java Programs." *SourceForge.net*. 21 Aug. 2009 <http://findbugs.sourceforge.net>.
3. Wheeler, David W. "FlawFinder." <www.dwheeler.com/flawfinder>.
4. Fortify Software, Inc. "Welcome to RATS – Rough Auditing Tool For Security." 2009 <www.fortifysoftware.com/security-resources/rats.jsp>.
5. Valgrind Developers. "Valgrind." <http://valgrind.org>.
6. GCC, the GNU Compiler Collection Wiki. "Mudflap Pointer Debugging." 10 Jan. 2008 <http://gcc.gnu.org/wiki/Mudflap_Pointer_Debugging>.
7. Roy, Graeme S. "mpatrol." 16 June 2009 <http://mpatrol.sourceforge.net>.
8. Wireshark Foundation. "About Wireshark." <www.wireshark.org/about.html>.
9. Wagner, David, and Bruce Schneier. *Analysis of the SSL 3.0 Protocol*. Proc. of the Second USENIX Workshop on Electronic Commerce. Oakland: 18-20 Nov. 1996. Paper Revised 15 Apr. 1997 <www.schneier.com/paper-ssl-revised.pdf>.
10. Bernstein, D.J. "SYN Cookies." <http://cr.yp.to/syncookies.html>.
11. Schönefeld, Marc. *Pentesting J2EE*. Proc. of Black Hat Federal 2006. 25 Jan. 2006 <www.blackhat.com/presentations/bh-federal-06/BH-Fed-06-Schoenefeld-up.pdf>.
12. Crosby, Scott A., and Dan S. Wallach. *Denial of Service via Algorithmic Complexity Attacks*. Proc. of the 12th USENIX Security Symposium. Washington, D.C.: 4-8 Aug. 2003 <www.cs.rice.edu/~scrosby/hash/CrosbyWallach_UsenixSec2003.pdf>.
13. Pal, Partha, Franklin Webber, and Richard Schantz. *The DPASA Survivable JBI—A High-Water Mark in Intrusion-Tolerant Systems*. Proc. of the 2nd EuroSys Workshop on Recent Advances in Intrusion-Tolerant Systems. Lisbon, Portugal: 23 Mar. 2007 <http://wraits07.di.fc.ul.pt/4. pdf>.

Note

[1] A unit of information consisting of payload (the information) and metadata describing the information and used for brokering.

A Look at “Software Security Engineering: A Guide for Project Managers”©

Julia H. Allen, Dr. Robert J. Ellison, and Dr. Nancy R. Mead
SEI

Sean Barnum and Dr. Gary McGraw
Cigital, Inc.

The goal of software security engineering is to build better, defect-free software. The book “Software Security Engineering: A Guide for Project Managers” [1][1]*—and its key resource, the Build Security In (BSI) Web site—provide software project managers with sound practices that they can evaluate and selectively adopt to help reshape their own development practices. Software developed and assembled using these practices should contain significantly fewer exploitable weaknesses.*

Software is ubiquitous. Many of the products, services, and processes that organizations use and offer are highly dependent on software to handle the sensitive and high-value data on which people’s privacy, livelihoods, and very lives depend. National security relies on increasingly complex, interconnected, software-intensive information systems—systems that (in many cases) use the Internet or Internet-exposed private networks as their means for communication and transporting data.

Dependence on IT makes software security a key element of business continuity, disaster recovery, incident response, and national security. Software vulnerabilities can jeopardize intellectual property, consumer trust, business operations and services, and a broad spectrum of critical applications and infrastructures, including everything from process control systems to commercial application products.

The integrity of critical digital assets (systems, networks, applications, and information) depends on the reliability and security of the software that enables and controls those assets. However, business leaders and informed consumers have growing concerns about the scarcity of practitioners with requisite competencies to address software security [2]. They have concerns about suppliers’ capabilities to build and deliver secure software that can be used with confidence and without fear of compromise. Application software is the primary gateway to sensitive information. According to the Deloitte survey of 169 major global financial institutions, current application software countermeasures are no longer adequate—application security is the number one issue for chief information officers [3].

The absence of security discipline in today’s software development practices often produces software with exploitable weaknesses. Security-enhanced processes and practices—and the skilled people to manage and perform them—are required to build software that can be trusted to operate more securely than the software being used today.

That being said, there is an economic counterargument, or at least the perception of one. Some business leaders and project managers believe that developing secure software slows the process and adds to the cost while not offering any apparent advantage. In many cases, when the decision reduces to *ship now* or *be secure and ship later*, ship now is almost always the choice made by those who control the money but have no idea of the risks. Information combating this argument—showing ways software security has led to cost and schedule reduction and documented successful experiences (e.g., Microsoft’s Security Development Lifecycle)—is out there.

> *“Security-enhanced processes and practices ... are required to build software that can be trusted to operate more securely than the software being used today.”*

The Goal of Software Security Engineering

Software security engineering is using practices, processes, tools, and techniques for addressing issues in every phase of the software development life cycle (SDLC). Software that is developed with security in mind is typically more resistant to both intentional attack and unintentional failures. One view of secure software is that software is engineered “so that it continues to function correctly under malicious attack” [4] and is able to recognize, resist, tolerate, and recover from events that intentionally threaten its dependability. Broader views that can overlap with software security (e.g., software safety, reliability, and fault tolerance) include proper functioning in the face of unintentional failures or accidents, inadvertent misuse and abuse, and software defect and weakness reduction to the greatest extent possible (regardless of its cause).

The goal of software security engineering is to build better, defect-free software. Software-intensive systems that are constructed using more securely developed software are better able to:

- Continue operating correctly in the presence of most attacks by either resisting the exploitation of weaknesses in the software or tolerating the failures that result from such exploits.
- Limit the damage from attack-triggered fault failures that the software was unable to resist—or tolerate and recover quickly from those failures.

Software Security Practices

No single practice offers a universal silver bullet for software security. With this in mind, “Software Security Engineering” provides software project managers with sound practices that they can evaluate and selectively adopt to help reshape their own development practices. The objective is to increase the security and dependability of the software produced by these practices, both during its development and its operation.

The book—and material referenced on the BSI Web site at <https://buildsecurityin.us-cert.gov>—identify and compare potential new practices that can be adapted to augment a project’s current software development practices. These resources also greatly increase the likelihood of producing more secure software and meeting specified security requirements. As one example, assurance cases can be used to assert and specify desired security properties, including the extent to which security practices have been successful in satisfying security requirements.

Software developed and assembled using software security practices should contain significantly fewer exploitable weaknesses. Such software can be relied on to more capably recognize, resist or tolerate, and recover from attacks—in turn functioning more securely in an operational environment. Project managers responsible for ensuring that software and systems adequately address their security requirements throughout the SDLC can review, select, and tailor guidance from the book and Web site as part of normal project management activities.

The five key takeaways from the book are as follows:

1. Software security is about more than eliminating vulnerabilities and conducting penetration tests. Project managers need to take a systematic approach to incorporate sound software security practices into their development processes. Examples include security requirements elicitation, attack pattern and misuse/abuse case definition, architectural risk analysis, secure coding and code analysis, and risk-based security testing.
2. Network security mechanisms and IT infrastructure security services do not sufficiently protect application software from security risks.
3. Software security initiatives should follow a risk management approach to identify priorities and what is good enough, understanding that software security risks will change throughout the life cycle. Risk management reviews and actions are conducted during each SDLC phase.
4. Developing secure software depends on understanding the operational context in which it will be used. This context includes conducting end-to-end analysis of cross-system work processes, working to contain and recover from failures using lessons learned from business continuity, and exploring failure analysis and mitigation to deal with system and system-of-systems complexity.
5. Project managers and software engineers need to think like an attacker in order to address the range of things that software should not do and how software can better resist, tolerate, and recover when under attack. The use of attack patterns and misuse/abuse cases throughout the SDLC encourages this perspective.

Practice Maturity and Relevance

As a community, we recognize that some software security practices are in broader use and thus more tested and mature than others, such as security coding practices and vulnerability testing. As a practice description and selection aid, descriptive tags mark the book's sections and key practices in two practical ways:

1. Identifying the content's relative *maturity of practice* as follows:
   - **Maturity Level 1 (L1):** The content provides guidance for how to think about a topic for which there is no proven or widely accepted approach. The intent of the description is to raise awareness and aid in thinking about the problem and candidate solutions. The content may also describe promising research results that may have been demonstrated in a constrained setting.
   - **Maturity Level 2 (L2):** The content describes practices that are in early pilot use and are demonstrating some successful results.
   - **Maturity Level 3 (L3):** The content describes practices that are in limited use in industry or government organizations, perhaps for a particular market sector.
   - **Maturity Level 4 (L4):** The content describes practices that have been successfully deployed and are in widespread use. These practices can be used with confidence. Experience reports and case studies are typically available.
2. Identifying the designated audiences for which each chapter section or practice is most relevant:
   - **E:** Executive and senior managers.
   - **M:** Project and mid-level managers.
   - **L:** Technical leaders, engineering managers, first-line managers, and supervisors.

> *"Software developed and assembled using software security practices should contain significantly fewer exploitable weaknesses ..."*

Build Security In: A Key Resource

Since 2004, the DHS Assurance Program has sponsored development for the BSI Web site, a significant resource used in developing "Software Security Engineering." BSI content, referenced throughout the book, is based on the principle that software security is fundamentally a software engineering problem and must be managed in a systematic way throughout the SDLC.

BSI both contains and links to a broad range of information about sound practices, tools, guidelines, rules, principles, and other knowledge to help project managers deploy software security practices and build secure and reliable software. Contributing authors to this book and articles appearing on the BSI site include senior staff from the SEI and Cigital, Inc.

Readers can consult BSI for additional details, ongoing research results, and information about related Web sites, books, and articles.

Start the Journey

As software and security professionals, we will never be able to get ahead of the game by addressing security solely as an operational issue. Attackers are creative, ingenious, and increasingly motivated by financial gain. They have been learning how to exploit software for several decades; the same is not true for software engineers, and we need to change this. Given the extent to which nations, economies, businesses, and families rely on software to sustain and improve the quality of life, we must make significant progress in putting higher quality and more secure software into production. The practices described in "Software Security Engineering" serve as a useful starting point.

Each project manager needs to carefully consider the knowledge, skills, and competencies of their development team, their organizational culture's tolerance (and attention span) for change, and the degree to which sponsoring executives have bought in (a prerequisite for sustaining any improvement initiative). In some cases, it may be best to start with secure software coding and testing practices: They are the most mature, have a fair level of automated support, and can demonstrate some early successes, providing visible benefits in helping software security efforts gain support and build momentum. On the other hand, secure software requirements engineering and architecture and design practices offer opportunities to address more substantive root cause issues early in the life cycle that, if left unaddressed, will show up in the code and test phase. Practice selection and tailoring are specific to each organization and project based on objectives, constraints, and the criticality of the software under development.

Project managers and software engineers need to develop a better understanding of what constitutes secure software—honing their skills to think like an attacker—applying this mindset throughout the SDLC. The book describes practices to get this ball rolling, such as attack patterns and assurance cases. Alternatively, if you have access to experienced security analysts, adding a few of them to your development team can get this jump-started.

Two of the key project management practices are 1) defining and deploying a risk management framework to help inform practice selection and determine where best to devote scarce resources and 2) identifying the best way to integrate software security practices into the organization's current SDLC.

Also keep in mind that this process, if done properly, will take time. As John Steven stated:

> Don't demand teams to begin conducting every activity on day one. Slowly introduce the simplest activities first, then iterate ... [Have] patience. It will take at least three to five years to create a working, evolving software security machine. Initial organization-wide successes can be shown within a year. Use that time to obtain more buy-in and a bigger budget. [5]

Clearly, there is no one-size-fits-all approach. Project managers and their teams need to think through the choices, define their tradeoff and decision criteria, learn as they go, and understand that this effort requires continuous refinement and improvement.

In Closing

Sound software security engineering practices should be incorporated throughout the entire SDLC. "Software Security Engineering" is one resource that captures both standard and emerging software security practices and explains why they are needed to develop more security-responsive and robust systems.◆

Software Defense Application

The book "Software Security Engineering: A Guide for Project Managers"—based primarily around the DHS' Build Security In Web site—is a defense software industry mainstay for selecting the right systems assurance tools. This article illuminates ways to use the book in planning software security improvements, from the early development stages through deployment and operations. As well, software developed using the suggested practices will result in on-time and on-budget projects that are more predictably secure.

References

1. Allen, Julia, et al. *Software Security Engineering: A Guide for Project Managers.* Upper Saddle River, NJ: Addison-Wesley, 2008.
2. Carey, Allan. "2006 Global Information Security Workforce Study." IDC. Oct. 2006 <www.isc2.org/uploadedFiles/Industry_Resources/workforcestudy06(1).pdf>.
3. Deloitte. *2007 Global Security Survey: The Shifting Security Paradigm.* Sept. 2007 <www.deloitte.com/assets/Dcom-Serbia/Local%20Assets/Documents/rs_Deloitte_Global_Security_Survey_2007.pdf>.
4. McGraw, Gary. *Software Security: Building Security In.* Boston: Addison-Wesley Professional, 2006.
5. Steven, John. "Adopting an Enterprise Software Security Framework." *IEEE Security & Privacy 4.2* (Mar./Apr. 2006): 85-87 <https://buildsecurityin.us-cert.gov/daisy/bsi/resources/published/series/bsi-ieee/568. html>.

Note

1. Material from this article has been taken from the preface and Chapter 8 of "Software Security Engineering: A Guide for Project Managers." It is reproduced with permission of Pearson Education, Inc. For additional information about the book, including a full table of contents, please refer to <www.informit.com/store/product.aspx?isbn=032150917X> and <www.sei.cmu.edu/library/abstracts/books/032150917X.cfm>. As well, podcasts including Julia Allen, Nancy Mead, and Gary McGraw are a nice introduction to the book's content. Find the CERT Podcast series at <www.cert.org/podcast/#softsecurity>.

Coming Events

March 9-12

12th Semi-Annual
Software Assurance Forum
McLean, VA
https://buildsecurityin.us-cert.gov/daisy/bsi/events.html

April 26-29

22nd Annual Systems and Software Technology Conference
Salt Lake City, UT
www.sstc-online.org

May 3-7

DISA Customer Partnership Conference 2010/AFCEA Technology Showcase
Nashville, TN
http://events.jspargo.com/disa10

May 10-14

PSQT 2010 West
Las Vegas, NV
www.psqtconference.com/2010west

May 24-27

Siemens PLM Connection 2010
Nashville, TN
http://event.plmworld.org

June 6-10

IBM Rational Software Conference
Orlando, FL
http://www-01.ibm.com/software/rational/innovate

June 6-11

Better Software Conference
Las Vegas, NV
www.sqe.com/BetterSoftwareConf

Coming Events: **Please submit coming events that are of interest to our readers at least 90 days before registration. E-mail announcements to: <marek.steed.ctr@hill.af.mil>.**

About the Authors

Julia H. Allen is a senior member of the technical staff within the CERT Program at the SEI. In addition to her work in software security and assurance, Allen conducts research in security governance, operational resilience, and metrics. She is the author of "The CERT Guide to System and Network Security Practices," and "Governing for Enterprise Security," and co-hosts discussions for the CERT Podcast Series: "Security for Business Leaders."

SEI
4500 Fifth AVE
Pittsburgh, PA 15213-3890
Phone: (412) 268-7700
E-mail: jha@sei.cmu.edu

Robert J. Ellison, Ph.D., is a member of the Survivable Systems Engineering Team within the CERT Program at the SEI, and has served in a number of technical and management roles. Ellison regularly participates in the evaluation of software architectures and contributes from the perspective of security and reliability measures. His research draws on that experience to integrate security issues into the overall architecture design process. Ellison's current work explores developing reasoning frameworks to help architects select and refine design tactics to mitigate the impact of a class of cyberattacks.

E-mail: ellison@sei.cmu.edu

Nancy R. Mead, Ph.D., is a senior member of the technical staff in the Survivable Systems Engineering Group within the CERT Program at the SEI. She is a faculty member in the Master of Software Engineering and Master of Information Systems Management programs at Carnegie Mellon University. Mead is a fellow for the IEEE and the IEEE Computer Society and is also a distinguished member of the Association for Computing Machinery. Mead has more than 100 publications and invited presentations.

E-mail: nrm@sei.cmu.edu

Sean Barnum is a principal consultant at Cigital and is technical lead for their federal services practice. He has more than 20 years of experience in the software industry in the areas of development, software quality assurance, quality management, process architecture and improvement, knowledge management, and security. He is involved in numerous knowledge standards-defining efforts, including Common Weakness Enumeration, Common Attack Pattern Enumeration and Classification, and other elements of software assurance programs for the DHS and DoD. He is also the lead technical subject matter expert for the Air Force Application Software Assurance Center of Excellence.

Cigital, Inc.
21351 Ridgetop CR, STE 400
Dulles, VA 20166
Phone: (703) 473-8262
E-mail: sbarnum@cigital.com

Gary McGraw, Ph.D., is the chief technical officer of Cigital, Inc. He is a globally recognized authority on software security and the author or co-author of six best-selling books on this topic. The latest, "Exploiting Online Games," was released in 2007. His other titles include "Java Security," "Building Secure Software," "Exploiting Software," and "Software Security"; he is also contributing editor for the Addison-Wesley Software Security series. McGraw has written more than 90 peer-reviewed scientific publications, and authors a monthly column for Dark Reading <www.darkreading.com>.

Phone: (703) 404-9293
E-mail: gem@cigital.com

Building Security In Using Continuous Integration

Thomas Stiehm and Gene Gotimer
Coveros

Building security into software is harder than it should be. This article explores a way to align application security practices with other software development best practices in order to make building security in easier to manage and more cost effective. In particular, this article looks at combining continuous integration (CI) with security testing and secure static code analysis.

Application development and security practices are often at odds. Application development is concerned with creating software quickly with the most features possible in the minimum amount of time. Application security is concerned with finding and removing security vulnerabilities and releasing software when critical security risks have been mitigated.

Many project stakeholders see application security practices as an increase in scope that adversely impacts the software delivery schedule. In order to build secure applications, it is important to align application development and security practices. Our analysis has found that one of the best ways to do that is to integrate secure code analysis and security testing into CI. CI is a software development practice where members of a team integrate their work frequently, verifying each integration by an automated build/test process to detect integration errors as quickly as possible [1]. Using CI, the time and effort to build security into the development process can be minimized, making teams more likely to include security practices in their software development process and thereby reducing the risk of a successful attack.

Immediate Notification

CI ensures that ongoing changes to the source code do not break the intent or design of the software. If a change does break the software, that break is identified immediately and can be fixed with a minimal cost and impact to the project's schedule.

CI started with the notion that each CI cycle should make a *clean build* from an up-to-date checkout from the source code repository, and that a set of unit tests should be run against the clean build as a regression against the changes in the code base. If the build and unit tests pass, then the recent checked-in changes did not break the software. If the build or unit tests fail, then the changes broke the software, and the CI server immediately notifies the team that the software is broken.

Secure Development

CI is now the foundation for serving many crucial software development process tasks. Originally focused on compiling and unit testing, CI practices have grown and evolved over time. They now include expanded practices, such as functional testing and code analysis to evaluate the health of a project. By integrating security testing and secure code analysis, CI can be further leveraged to include secure development practices while minimizing the amount of extra effort required to get the benefits of secure development. Since it is tied to CI, security testing and secure code review begins when a project begins and runs continuously throughout project development. With CI, security vulnerabilities testing becomes part of the regression test bed, executed automatically with each successive build on the CI platform.

Changing Testing Economics

Technology advances have changed the economics of testing, allowing more aggressive approaches to testing than historically possible. Using CI for build, test, and analysis automation has increased the depth and breadth of tests while also making them faster and less expensive. By making it cheap and easy to perform tests, teams are encouraged to test more and test sooner in the development cycle, reducing the cost of fixing bugs. It has also made it easier for managers and non-technical stakeholders to understand project progress and health.

For many projects, testing has gone from a process that slowed deployment down to one that provides true quality assurance, in turn helping stakeholders have more confidence in their projects. By reducing the cost of many quality control aspects of application development, teams have been able to use those controls more often and more effectively and have accelerated development while improving quality. This same change can be applied to security practices integrated into CI. The reduced cost of gathering security vulnerability data will encourage teams to collect the data more often and sooner in their development cycle, reducing the cost to fix issues such as cross-site scripting and SQL injection.

Building Security Testing Into CI

In order to integrate static code analysis and security testing into CI, a few key pieces of software are needed. Organizations should select a specific application for each of the software categories (as shown in Table 1), bearing in mind that some products might fit into more than one category. For example, Microsoft's Team Foundation Server (TFS) [2], fits into the CI Server, Source Code Repository, and Issue Tracking categories.

Choosing the Right Tools

It is possible to build a completely open source CI process that includes static code analysis and security testing. SecureCI is one example of a CI product that is made of all open source tools and can be downloaded and used free of charge [3]. There are also many good commercial products that, in some cases, provide more value than their open source alternatives, such as the Web crawling capability of AppScan and WebInspect.

There are many tools to choose from, ranging from unsupported to company-supported open source software to commercial software that is developed and maintained completely by the company that created it. In other words, an organization can purchase support contracts to commercial software—products that are then developed and maintained completely by the company that created them. There are even some commercial software products that are free to use but come with some restrictions, such as a limited number of users. There is no correct or incorrect tool as long as the acquired software works together and can meet organizational objectives.

Tool Integration

In general, the CI server will be the integration hub and orchestrator for a CI process. CI servers come with integration application programming interfaces (APIs) for many of the tools an organization puts in its toolset so that working with those tools will be easy. If there isn't an API for one of the selected tools, CI servers can be extended using scripting languages and compiled code plug-ins. This flexibility means that an organization can add any tools that can be scripted or programmatically controlled into their CI process, making it possible to add security practices to an existing CI process without having to reinvent that process. For example, with Hudson, in order to use PMD [4] or FindBugs [5] for static code analysis, all that is needed is creation of a build job that uses an Ant [6] build to run the tool. You then point the tool's Hudson [7] plug-in to the XML reports created during the Ant build. The plug-in picks up the reports and parses them for display using HTML and generated graphics.

The CI server market is full of both open source and commercial products. Many of the open source CI servers have commercial support contracts available. Because of the mature CI server market, there is no advantage in using either open source or commercial products. The exception to this statement is when an organization is considering development ecosystem suites like Microsoft's TFS and Visual Studio suite.

Integrating security testing tools requires a little more work. The application under test needs to be deployed and running because security testing tools work by interacting with the application, analyzing the requests and responses from the point of view of a Web browser. This means that the CI server will have to deploy to an application server, start it, and then kick off the security testing tool. While it seems like a lot of work, there are a number of good examples on the Internet to help get started. The commercial security testing tools can be configured to just launch and crawl an application.

Creating Multiple Complementary Builds to Support Specific Needs

One of the first things organizations will notice about using static code analysis and automated security testing tools is that build times—the time it takes to go through a CI process—will increase significantly. Because of this increase, it is common for projects to use multiple CI jobs. The different jobs are set to run on different intervals and for different reasons. For instance, most teams have a quick job that runs within 10 minutes of a check-in and produces a result within 10 minutes. This quick job consists of a clean build and only executes unit tests. This quick build tells the developers that the new code works and that all of the unit tests pass (i.e., the code hasn't introduced defects into previously working code). Many teams will also use longer running tests (a couple of hours), compiling the project and running the unit tests, while also executing other kinds of tests (e.g., database tests and automated functional tests). These tests take longer to complete and have a longer feedback cycle. By executing multiple jobs, an organization can provide the team with feedback as quickly as possible for a given type of feedback. Finally, many projects have jobs running from once a day to once a week that perform static analysis or security testing. This allows the processes to run at their own pace without slowing down other test processes.

The selection of a source code repository is generally based on what already exists in an organization's environment. From the overall goal of building security into applications, the choice of source repository makes little difference—it just needs to work with the CI server.

Utilizing Both Commercial and Open Source Tools

There is a healthy marketplace for issue tracking applications having both open source and commercial products.

Commercial issue tracking software has an advantage over open source in terms of the reporting and integration options. Commercial applications tend to have a better and more customizable reporting system and tend to integrate more with (usually commercial) software that might be used in an overall development process. That said, many projects and organizations don't need or use the extra capability of commercial issue tracking software. For small project teams or small companies, open source issue tracking software works just fine. For large enterprises with multiple related development projects, a commercial issue tracking application may offer needed features that can justify the cost of acquiring the software.

Open source unit testing tools are usually frameworks that provide a core unit testing capability. These tools require a developer to write and maintain unit tests. If the developer follows the conventions of the framework, running unit tests is very simple and easy and many CI servers can read and report on the results. The commercial products in the space build on top of the open source tools by adding the capability to generate unit tests. The commercial tools will scan the source code and determine how to exercise the code paths (in the source code) in order to get 100 percent test coverage and possibly add negative unit tests.

We have worked on projects that have used both developer-written unit tests and tool-generated unit tests. When dealing with security features, it is important to write comprehensive unit tests that exercise both the positive and negative paths

Table 1: *Tools for Different Software Categories*

Software Category	Description	Open Source Tools	Commercial Tools
CI Server	Server software that monitors the source code repository and runs the build when changes to the repository are detected.	Hudson, CruiseControl	TeamCity, Bamboo, TFS
Source Code Repository	Software that keeps source code, maintaining versions of the source files and file groups (i.e., labels and tags).	Concurrent Versions Systems, Subversion	Polytron Version Control System, Clearview, TFS
Issue Tracking	Software that is used to manage software issues and report their status.	Trac and Sonar, Bugzilla	Quality Center, TFS
Unit Testing	Tools or frameworks used by developers to test at a source code level.	JUnit, NUnit	JTest
Functional Testing	Tools, frameworks, or applications used to test the functionality of software.	Selenium, Watir	Quick Test Professional, SilkTest
Security Testing	Tools, frameworks, or applications used to test the security aspects of software (i.e., penetration testing tools).	RatProxy, WebScarab	AppScan, WebInspect
Static Code Analysis	Tools, frameworks, or applications used to inspect either source code or compiled files for known issues.	FindBugs, PMD, CheckStyle	Fortify Source Code Analyzer

Software Defense Application

Application security is a priority for DoD applications, given the always on, global nature of the DoD mission. At the same time, it is important to make application security work within the development practices in common use among DoD development teams. By integrating application security practices with CI, we can address both the security needs of the applications as well as the efficiency and cost-effective requirements of the development teams.

through the code. Many security defects come from security features that don't have a proper *fail-safe* method for when there are program or data processing errors. Negative testing—or the testing for failure paths in the source code—addresses the correctness of failure states and can show that a specific security feature can fail-safely when something unexpected happens. If there is a large legacy code base that doesn't have unit tests, using a tool to generate unit tests is a good way to add tests quickly. Keep in mind, however, that the project team still needs to review all of those tests, change some, remove others, and write new tests covering situations that the tools couldn't.

Security Testing Tools

Security testing is an excellent way to discover many of the security vulnerabilities in an application. These tools are run against an application, usually in a testing environment, in its production configuration. One goal of this testing is to determine if the application has defects, making it vulnerable to outside attack while in production; another is to see if the application will fail safely when attacked. Failing safe has different meanings to different applications, with the basic guideline that an application should not give users unauthorized information or allow them to take unauthorized actions.

In order to automate many open source security-testing tools, they will need to be used in conjunction with functional test tools. For example Ratproxy [8], a popular open source security-testing tool, can't (unlike popular commercial security testing tools) crawl a Web application. This means that people using Ratproxy need to use another tool to crawl the Web application while Ratproxy is running. Another alternative is to have the testers use Ratproxy while they are conducting manual functional tests.

Another area where commercial security testing tools have an advantage is in reporting. Commercial security testing tools, like AppScan [9], have a customizable reporting capability that gives users numerous ways to report their findings (in order to conform to organizational standards or highlight different aspects of the findings for different audiences). The reporting capability also does a good job of explaining the findings—that is, how they can be exploited and remediated. In contrast, Ratproxy has only one report format that provides little detailed information about an issue beyond its name and where it was found. An analyst using Ratproxy has to figure out what the issue means and how to remediate it. Finding out this information isn't hard, but it can be time-consuming.

Static Code Analysis Tools

Static code analysis tools examine the code base for many problems including security and code style issues, potential code defects, and race conditions. These tools are more akin to automated code review than to unit and integration tests. The difference between commercial and static code analysis tools mirror the differences found in security testing tools: Commercial tools have better reporting capabilities and provide more information on the nature of the findings and suggested remediation. Another area where some commercial tools have added value is in the depth of analysis. Fortify Software's Source Code Analyzer [10] creates an entire model of the application under analysis, examining the data and control flow of the software to determine if there is a larger-context problem. Most open source static code analysis tools only look for problems in a local context (i.e., the immediate line of code, code block, or file).

Static code analysis can either be performed on the source code of an application or on the compiled binaries. For example, FindBugs runs against compiled Java class files. It can find bad practices, null pointer dereferences, static use of non-thread safe code, and security issues. PMD runs against Java source files. It can find dead code, performance issues, style issues, and potentially dangerous code practices. While there is overlap with FindBugs, using both is not considered redundant.

Data Analysis and Evaluation

Once the vulnerability data has been collected from both security testing and code analysis, it has to be analyzed and evaluated. The goal of this step is to decide if the findings are truly a problem and what to do about it. This step includes determining the priority and severity of the security issues and putting them into an issue tracking system. In order for application security practices to positively affect the project, issues uncovered in testing and analysis need to be tracked and fixed. By integrating security practices into CI, security issues are discovered and dealt with more quickly, in turn preventing many security risks from entering production and minimizing the possibility of exploitation.

Building application security practices into a project can be done with minimum impact to a project's budget, schedule, and resources. Integrating security testing and secure code analysis into CI is the first step to building security into software. These practices help build security awareness by showing developers how and why their code is vulnerable. They give testers the tools needed to find many security vulnerabilities, and project managers a way to demonstrate the results of security practices. While it would be instructive to provide quantitative analysis of the benefits of integration CI and security, the practice is still new enough that no major studies have been conducted.

Application security practices are hard for many teams to adopt because they don't have the time, budget, or resources needed. CI can help change the economics of security testing and analysis by giving project teams tools that can be deployed in their environment. These solutions can enable a team to quickly go from not considering security to a solid initial step in finding and proactively fixing security vulnerabilities.◆

References

1. Duvall, Paul, and Steve Matyas. *Continuous Integration: Improving Software Quality and Reducing Risk*. New York: Addison-Wesley, 2007.
2. Microsoft. "Team Foundation Server Home." *Team Foundation Server Home*. 2009 <http://msdn.microsoft.com/en-us/teamsystem/dd408382.aspx>.
3. Coveros. *Coveros: Research & Insights – Free Secure CI Download*. Fairfax, VA: Coveros, 2009.
4. InfoEther. "PMD" *SourceForge.net*. 2009 <http://pmd.sourceforge.net>.
5. The University of Maryland. "FindBugs – Find Bugs in Java Programs." *SourceForge.net*. 21 Aug. 2009 <http://findbugs.sourceforge.net>.
6. The Apache Ant Project. "Apache Ant – Welcome." *Apache Ant*. 15 Oct. 2009

About the Authors

Thomas Stiehm has been developing applications and managing the software development teams for 16 years. As Chief Technical Officer of Coveros, he is responsible for the oversight of all technical projects and integrating application security practices into software development projects.

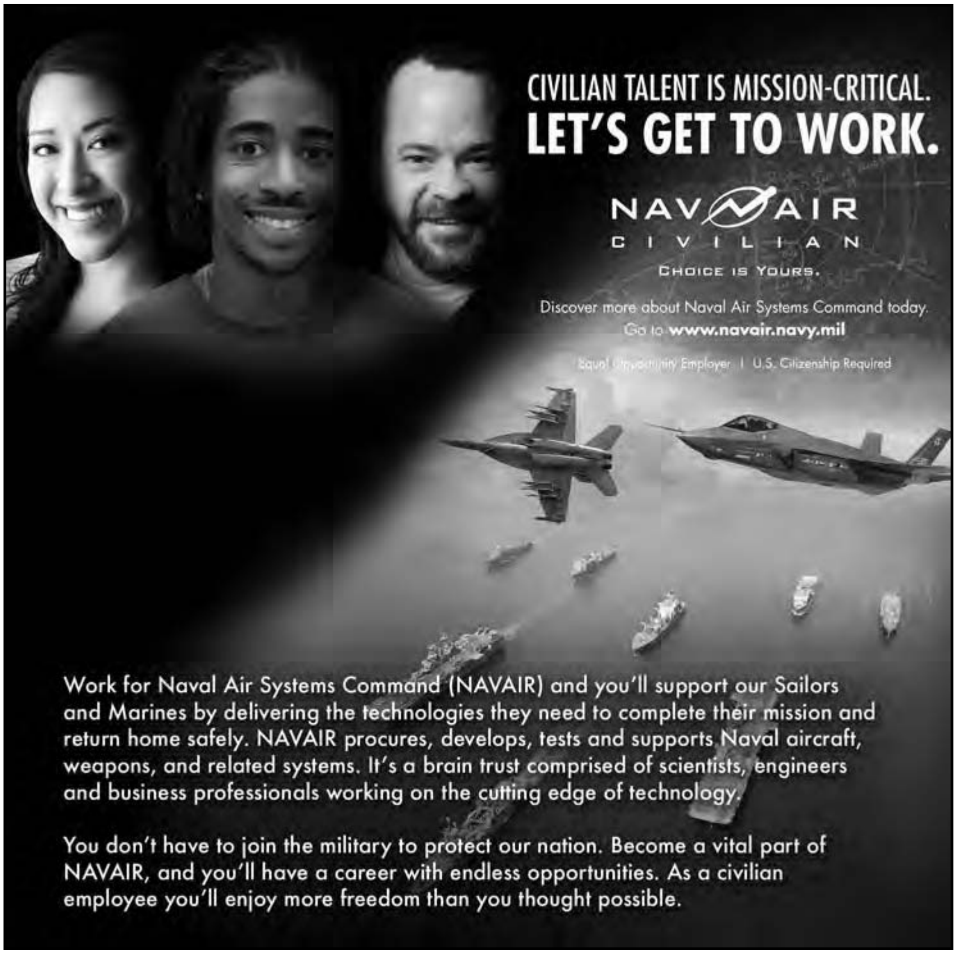
Coveros
4000 Legato RD
STE 1100
Fairfax, VA 22033
Phone: (703) 599-6243
E-mail: tom.stiehm@coveros.com

Gene Gotimer has been building Web applications and working with security for the last 13 years. He specializes in CI and Agile Java development.

Coveros
4000 Legato RD
STE 1100
Fairfax, VA 22033
Phone: (703) 963-1620
E-mail: gene.gotimer @coveros.com

<http://ant.apache.org>.

7. Sun Microsystems. "Hudson CI". *Hudson: Extensible continuous integration server*. 2009 <http://hudson-ci.org>.
8. Google. "Ratproxy: passive web application security assessment tool." Ratproxy – Project Hosting on Google Code. 2009 <http://code.google.com/p/ratproxy>.
9. IBM. "Help ensure Web site security and compliance." *IBM Web Site Security and Compliance – Rational*. 2009 <http://www-01.ibm.com/software/rational/offerings/websecurity>.
10. Fortify Software, Inc. "Source Code Analyzer (SCA) in Development." 2009 <www.fortify.com/products/detect/in_development.jsp>.



Web Sites

Making Security Measurable

http://measurablesecurity.mitre.org

At the heart of this issue's article *Systems Assurance as a Team Sport* are several MITRE-led information security community standardization activities and initiatives—now visit the Web site which houses comprehensive information on them all. The goal of this collaboration with government, industry, and academic stakeholders is to improve the measurability of security through enumerating baseline security data, providing standardized languages as a means of communicating accurately, and encouraging the sharing of the information with users by developing repositories. These efforts are helping to make security more measurable by defining the concepts that need to be measured and providing a location for high-fidelity communications about the measurements.

Engineering for System Assurance

www.acq.osd.mil/sse/docs/SA-Guidebook-v1-Oct2008.pdf

After reading *A DoD-Oriented Introduction to the NDIA's System Assurance Guidebook*, why not go directly to the source? This Guidebook provides process and technology guidance to increase the level of systems assurance. It is intended primarily to aid program managers and systems engineers who are seeking guidance on how to incorporate assurance measures into their system life cycles. The Guidebook discusses systems assurance by focusing on the entire system, and specifically addressing the assurance of security properties throughout the system life cycle. It also identifies processes, methods, techniques, activities, and tools for systems assurance, and focuses on the electronic hardware, firmware, and software elements that make up the system.

Forge.mil

www.forge.mil

In their article, Michael Atighetchi and Dr. Joseph Loyall state their belief that successful survivability assessment includes community-based collaboration platforms. They mention Forge.mil, a Defense Information Systems Agency-led family of services provided to support the DoD's technology development community. The system currently enables the collaborative development and use of open source and DoD community source software. These initial software development capabilities are growing to support the full system life cycle and allow for continuous collaboration among all stakeholders, including developers, testers, certifiers, operators, and users. Forge.mil's goals include enabling and promoting: cross-program sharing of software, system components, and services; early and continuous stakeholder collaboration; the rapid delivery of effective and efficient development and test capabilities for DoD technology development efforts; and the protection of the operational environment from potentially harmful systems and services.

Software Assurance (SwA)

https://buildsecurityin.us-cert.gov/swa

This issue's focus on systems assurance—and the article from Julia H. Allen, Dr. Robert J. Ellison, Dr. Nancy R. Mead, Sean Barnum, and Dr. Gary McGraw outlining the importance of the DHS' Build Security In (BSI) Web site—make it a perfect time to visit this BSI-sponsored site, specifically focused on SwA. Visitors can learn more about what SwA is, why it is critical, how it is advancing, the DHS' SwA Program, past and present SwA Forum meetings, and the progress of specific SwA "working groups." There are also links to several of their published resources, past and future events, as well as to Webinars, Webcasts, and podcasts.

Software Testing Boot Camp – The Economics of Testing

www.riceconsulting.com/public_pdf/STBC-WM.pdf

In *Building Security In Using Continuous Integration*, Thomas Stiehm and Gene Gotimer explore the economics of software testing, stating that it has "gone from a process that slowed deployment down to one that provides true quality assurance." This document has more evidence to support this assertion, discussing early defect detection and exploring why software defects are costly to find and fix. As part of a four-day "boot camp," it explores the cost of software defects, where defects originate, why a "big bang" testing phase at the end of a project doesn't work, the relative cost of defect fixing, as well as the benefits of testing early—and throughout a project.

Systems Assurance and Cybersecurity: Guidance and Tools

www.acq.osd.mil/sse/pg/guidance.html#sa

Looking for a "one-stop shop" for pertinent government and DoD documents on systems assurance? At this Systems and Software Engineering Web site—sponsored by the Office of the Deputy Under Secretary of Defense for Acquisition, Technology, and Logistics—visitors can access the most pertinent government publications and Web sites for systems assurance, including the "Defense Acquisition Guidebook," the historical "Acquisition Systems Protection Program" manual, Version 1.0 of "Engineering for System Assurance," and the recently-released "Acquisition Security Related Policies and Issuances Tool." This site also provides resources in the DoD focus areas of Defense Acquisition, Systems Engineering, Developmental Test and Evaluation, and Modeling and Simulation, as well as procedural documents from the Army, Navy, Air Force, and NASA.

Maturity Framework for Assuring Resiliency Under Stress

https://buildsecurityin.us-cert.gov/daisy/bsi/1016-BSI.html

Don O'Neill expands on his Sept./Oct. 2009 CrossTalk article *Meeting the Challenge of Assuring Resiliency Under Stress*, now exploring his Maturity Framework. The concept is intended to drive the business case and enterprise commitment towards the assurance of software security, business continuity, system survivability, and system of systems resiliency. The framework serves as a road map to help software developers and managers in understanding the software security decision process, to guide the selection of models appropriate for the enterprise, and to assist in their instantiation using local factors that best characterize the business environment and enterprise culture in achieving the best possible outcome.

Coming in the May/June Issue

Software Human Capital

Understanding of the economic importance of people goes back to the 1700s, with Adam Smith's declaration that humans are one of the four types of "capital." Software managers are quickly finding that success comes from human intelligence, innovation, and teamwork—and that failures are rooted in the sociological, rather than the technological.

This edition of CrossTalk documents success stories and explores the possibilities in developing the skills, talents, knowledge, and abilities needed to both perform at a high level and produce innovative products.

Look for it in your mailbox early May!

Issue sponsored by:

Department of Defense
Systems Engineering

Letter to the Editor

Dear CrossTalk Editor,

I have found CrossTalk to be an exceptional resource. In general, CrossTalk is a vast repository of knowledge for those in defense software engineering. Virtually every software topic has its own theme issue. One of our software engineers asked me what I knew about Agile Software Development. I immediately directed him to your April 2007 issue, themed *Agile Development*, which was easy to access on your Web site <www.stsc.hill.af.mil/crosstalk/2007/04/>.

Recent editions have also been both useful and enlightening. I really enjoyed CrossTalk's November/December 2009 issue, focused on 21st Century Defense:

- *PKI: The DoD's Critical Supporting Infrastructure for Information Assurance* (IA) by Susan Chandler and Jerrod Loyless was a great "101"-type article on Public Key Infrastructure (PKI) and IA. It was the first time I have really understood the mechanics of both, as well as how the PKI interfaces with my Common Access Card. I now appreciate IA a whole lot more, and what goes into making it work.
- The link to the DoD IA Certification and Accreditation Process page (see <www.stsc.hill.af.mil/crosstalk/2009/11/0911WebSites.html>) is handy to have around when IA issues come up. CrossTalk's *Web Sites* section is an extremely valuable and relevant resource for busy managers who don't have the time to ferret out "nuggets" like this.

I also liked the *Process Replication*-themed July/August 2009 issue:

- David P. Quinn's *Resistance as a Learning Opportunity* discussed why and how people resist process improvement (PI), and effectively showed how leaders can deal with resistance. It is often difficult for someone who is passionate about PI—and whose full-time job it is to help implement it—to understand and deal with those who are not. Quinn provided solid, practical suggestions for doing so. Just like a salesperson making a sale, PI folks need to "sell" PI with what's in it for the "resisters." It's a fine article that helps the PI movement go forward.
- Kudos to Gary Petersen for his BackTalk article, *Raiders of the Lost Art.* Long ago, I learned the difference between "efficient" and "effective." Something that is fast and cheap may be efficient, but not effective. I read management book after management book that tried to teach people the old ways of communication pointed out in Petersen's article. I don't know if we can ever get the toothpaste back in the tube, or the genie back in the bottle, but thank you very much, Gary, for pointing out the wrong direction we are going with communication and offering good suggestions for getting society back on course.

Thank you, CrossTalk, for your diligence, dedication, and great articles!

—Al Kaniss
Naval Air Systems Command
alan.kaniss@navy.mil

My Own Kind of UML: Innovative Technology Transfer With the Universal Motion Language

I've been thinking a lot about why it is so hard to get software right, and I think I may have hit on something.

I have noticed when visiting with relatives who live in the country that they are mechanically astute. They can fix or jury-rig almost anything. They work on problems at least as complicated as, say, running an automated payroll system for hourly employees. Have *you* ever tried to replace the boot on a compensating velocity gear on a tractor, or unsnarl an old fishing line reel and not lose that eight-pound bass? See what I mean!

What is of particular interest to me is their multi-channel knowledge transfer methodology. When describing a technique or process, conversations are punctuated and animated with hand motions and body postures. I get tired just watching one explain how to tighten a strand of barbed wire with nothing but a broomstick and some seagrass string, or how to pluck a chicken for supper on the kitchen table when it is storming outside.

You can look all around and see different groups with their own lexicon of hand and body motions: baseball coaches instructing a runner to steal, steelworkers guiding crane operators as they put I-beams into place, or a driver expressing disdain—with merely one finger—when another driver cuts them off.

Software developers are no less creative and resourceful, but are clearly missing the hand motions and body postures. I recommend computer science centers of excellence immediately pair up with psychologists in academia and begin developing these visual clues forthwith.

As a departmental process expert, it is my job to introduce and explain the latest processes and forms required for all projects. In my opinion, visuals are needed.

I have worked up a few on a trial basis, such as inserting a new item in the middle of a doubly linked list. I find the hand motions needed to help explain how to create a new relational table to be much less complicated than the doubly linked list—though initial trials with my colleagues on the job have so far proved unproductive.

I have also attempted the utilization of several hand signs, such as illuminating the walk-through process using two fingers walking across the desk. Some of the trainees responded with their own lower energy-state hand sign using less motion ... and fewer fingers. Hand and body motions are often more efficient (requiring less mental effort) than words in expressing concrete ideas and procedures. It is easier to describe a spiral staircase with your hands than it is with words.

In my humble opinion, these less than enthusiastic reactions are a case of new technology being rejected by seasoned veterans who fail to see the value in something they didn't invent themselves. It is, as they say, "not the way we do things around here."

I wanted to call this new technology UML, Universal Motion Language, but I am told that acronym has already been taken by some upstart language those smarty college new-hires keep wanting us to use[1]. I also toyed with calling it Associative Symbolic Signing, but my wife nixed that name. She suggested Symbolic Transfer Using Polite Indicative Directions.

The need for this new technology is accelerating faster than Moore's Law[2] as new ideas for improving software development productivity are popping up, I suspect, at roughly two new ideas every 18 months. Concomitant with the new ideas are their techniques, processes, models, books, and conferences, thus driving up the need for better knowledge transfer techniques exponentially.

We are almost to the point where we need both an XML open gesture tag and close gesture tag to identify the ever-growing list of standard gestures and distinguish them from other more prosaic hand motions and postures, such as combing one's hair, scratching one's posterior, or testing the pH of one's azalea bed.

Adapting this technology for supply chain automation activities should be a great place to start since the hand signals for driving a truck, pulling inventory, and shelf restocking are fairly intuitive, while request-for-price and backorder transactions would be more problematic.

But if we make this technology "freeware" and also give away rather than sell the documentation, then the open source and freeware communities can make short work of those type problems. After all, no problem can withstand 10,000 hands.

—Carl Wayne Hardeman
cwhardeman@yahoo.com

Notes

[1] They also suggested adopting a technology they call PowerPoint. We told them that is not the way we do things around here.

[2] Moore's Law, which dates back to 1965, states that the number of transistors on a chip will double every two years. See <http://en.wikipedia.org/wiki/Moore's_law> to learn more.

Can You BackTalk?

Here is your chance to make your point without your boss censoring your writing. In addition to accepting articles that relate to software engineering for publication in CrossTalk, we also accept articles for the BackTalk column. These articles should provide a concise, clever, humorous, and insightful perspective on the software engineering profession or industry or a portion of it. Your BackTalk article should be entertaining and clever or original in concept, design, or delivery, and should not exceed 750 words.

For more information on how to submit your BackTalk article, go to <www.stsc.hill.af.mil>.

CROSSTALK / 517 SMXS/MXDEA
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

Homeland Security

Software Assurance

Software is essential to enabling the nation's critical infrastructure. To ensure the integrity of that infrastructure, the software that controls and operates it must be secure and resilient.

Software Assurance Community Resources and Information Clearinghouse provides collaboratively developed resources. Learn more about relevant programs and how you can become involved.

https://buildsecurityin.us-cert.gov/swa/

Security must be "built-in" and supported throughout the lifecycle. Visit https://buildsecurityin.us-cert.gov to learn more about the practices for developing and delivering software to provide the requisite assurance. Sign up to become a free subscriber and receive notices of updates.

The Department of Homeland Security provides the public-private collaboration framework for shifting the paradigm to software assurance.

CROSSTALK thanks the above organizations for providing their support.